ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA
- Sede di Forlì -

CORSO DI LAUREA
IN INGEGNERIA AEROSPAZIALE
Classe: L-9

ELABORATO FINALE DI LAUREA

in

Impianti Aerospaziali

PROGETTO E REALIZZAZIONE DI UN DISPOSITIVO DI CONTROLLO E GESTIONE REMOTO PER UN SISTEMA DI TRACKING SATELLITARE (ALMATRACKER)

CANDIDATO

Igor Gai

RELATORE

Prof. Paolo Tortora

CORRELATORE

Ing. Claudio Bianchi

Anno Accademico 2012/2013
Sessione II[a]

Sommario

Indice delle figure

Indice delle tabelle

Capitolo Primo

INTRODUZIONE

La seguente tesi presenta lo sviluppo di un sistema di controllo e gestione remota per il tracking di un satellite. Il progetto, denominato ALMATracker, è sviluppato dal corso di Ingegneria Aerospaziale della scuola di Ingegneria e Architettura Aerospaziale dell'Università di Bologna con sede a Forlì. Consiste nella creazione di una motorizzazione per antenne su due assi, movimentata da un hardware commerciale programmabile. Il posizionamento può essere eseguito sia manualmente, su richiesta di un utente da PC remoto, sia automaticamente secondo un'orbita preimpostata. I setpoint di velocità o posizione sono elaborati dal sistema fino ad ottenere un segnale che procede alla movimentazione in velocità dell'antenna. Il comando automatico, invece, orienta l'antenna in modo tale da mantenerla fissa su una traiettoria orbitale di uno specifico spacecraft. La movimentazione automatica segue funzioni polinomiali fornite dall'utente, ricavate da software di propagazione e predizione esterno al sistema ALMATracker. In questo caso il sistema deve procedere alla rotazione mantenendo la velocità richiesta dalla funzione polinomiale. Il controllo effettuato in catena chiusa è attuato tramite una serie di trasduttori di posizione presenti nel sistema.

1.1. PROGETTO ALMATRACKER

Il progetto ALMATracker consiste nella creazione di una motorizzazione di antenna su due assi per il tracking satellitare la cui realizzazione si divide in tre macro-blocchi principali:

- un sistema meccanico costituito da antenna e motorizzazione;
- un sistema hardware costituito da CompactRIO e PC;
- un sistema software costituito dagli algoritmi di movimentazione e controllo del sistema.

Tutto il sistema di movimentazione deve essere chiuso in cicli di retroazione (in the loop), cioè in un sistema hardware-software che permetta non solo di comandare la movimentazione ma anche di controllare lo spostamento effettivamente avvenuto, ed adotti eventualmente una correzione automatica della traiettoria di inseguimento. Il sistema di controllo deve interagire con quello di movimentazione in maniera attiva, utilizzando il segnale di ritorno dai sensori per verificare l'efficacia delle azioni intraprese. Nei prossimi paragrafi verranno approfonditi i punti principali di cui è composto il software, e di come questo è stato sviluppato per giungere all'efficienza più alta con i componenti a disposizione.

Capitolo Secondo

COMPONENTI DEL SISTEMA

Il sistema ALMATracker è composto di una serie di dispositivi per l'acquisizione degli angoli caratteristici, il controllo degli azionamenti e la gestione logica del sistema intero.

Il nodo centrale del sistema è costituito da un'elettronica programmabile della NI, la cRIO. Ad essa sono collegati tutti gli attuatori e tutti i trasduttori del sistema. Il controllo da parte dell'utente è eseguito non direttamente sulla cRio ma da PC remoto. Questa soluzione è stata adottata per rendere il più possibile remotata e autonoma la stazione di tracking.

2.1. PC CLIENT

L'HID del sistema ALMATracker è costituito da un PC Windows. Il PC rappresenta un terminale sul quale è eseguita quella parte del software che permette all'utente di potersi collegare al sistema remoto e di poterne controllare modalità di esecuzione del tracking – manuale o automatica – e modificarne i parametri (ad esempio i setpoint o le polinomiali). Il carico di calcolo richiesto al PC è minimo poiché le sue funzioni si limitano a visualizzazione, modifica dei parametri e comunicazione in TCP con la stazione di tracking.

2.2. COMPACT RIO

La CompactRIO (d'ora in poi cRIO) è un sistema hardware integrato per l'acquisizione e l'elaborazione dati commercializzato dalla National Instuments. Questo hardware è costituito da un controller, uno chassis e uno o più moduli di acquisizione inseribili nello chassis. Il controller, che ospita l'unità di elaborazione RT (Real-Time), è direttamente collegato allo chassis. All'interno dello chassis è alloggiato il chip FPGA collegato tramite un bus, sia ai vari moduli estraibili, sia al controller RT. Il funzionamento del sistema può avvenire o direttamente con una

comunicazione tra RT e i vari moduli alloggiati nello chassis chiamati C-module, oppure attraverso la programmazione diretta dell’FPGA nel caso siano richieste performance in termini di sincronia e velocità superiori. Sia il controller RT sia l’FPGA sono programmati in linguaggio LabVIEW con strumenti simili e medesima strategia software ma con diverse limitazioni. Nel caso dell’RT non si ha accesso diretto ai moduli e le frequenze di lavoro sono inferiori. Nel caso dell’FPGA si ha un’elevata frequenza di esecuzione ma con limitazioni all’uso di numeri interi avendo a disposizione operazioni binarie e non in virgola mobile. Nello specifico del progetto si è utilizzato una ***cRIO 9014*** con chassis ***NI 9103*** ed un unico C-module per l’acquisizione di segnali digitali ***NI 9401*** a 8 canali. I canali da DIO0 a DIO3 sono stati impostati in lettura, mentre quelli da DIO4 a DIO7 in scrittura.

Figura 1 – Elementi costituenti di una cRIO. Da sinistra si possono osservare: il modulo RT NI 9014, lo chassis a 4 slot ed infine il modulo di acquisizione dei segnali NI 9104.

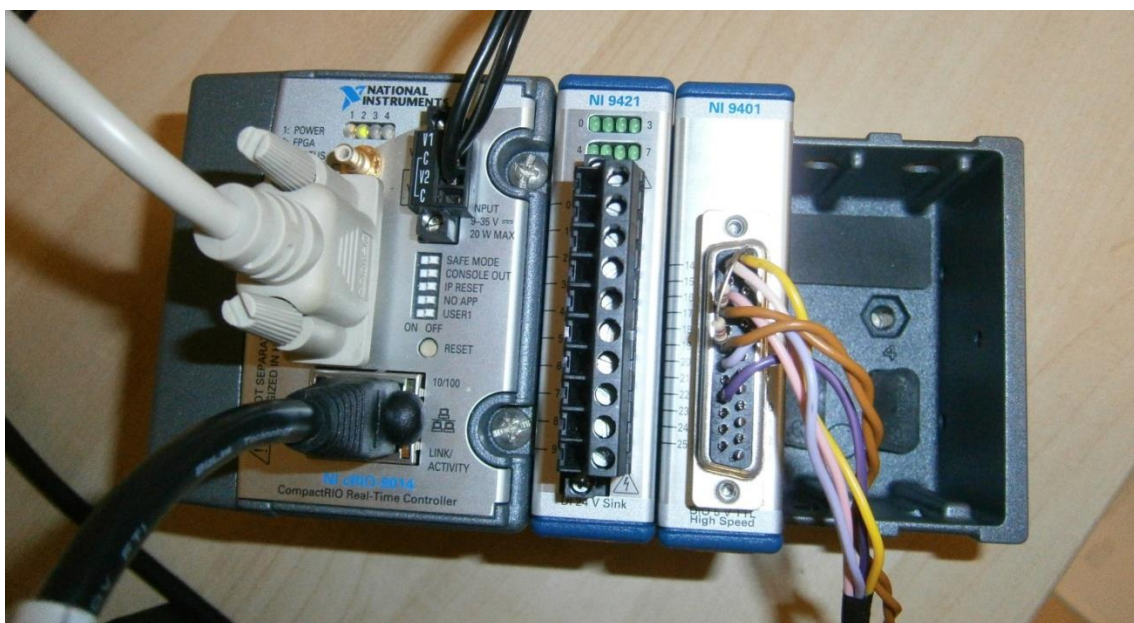



Figura 2 – Sistema cRIO durante l’esecuzione del software completo.

2.3. *ENCODER*

Gli encoder rotativi sono dei traduttori di posizione angolare che permettono la misura di un angolo tramite la conversione in un segnale elettrico digitale. Gli encoder sono tutti costituiti da un corpo fisso e da un rotore libero di ruotare, e si dividono in vari gruppi diversi per funzionamento e/o informazioni restituite:

A. principio di trasduzione
    a. capacitivo;
    b. magnetico;
    c. potenziometrico.
B. informazioni sull’angolo
    a. encoder tachimetrico;
    b. encoder relativo;
    c. encoder assoluto.

Il principio di trasduzione riguarda l’elettronica interna del dispositivo e non è importante ai fini della rilevazione. Più interessante è invece la classificazione secondo le informazioni sull’angolo, poiché una stessa posizione è identificata da codici diversi a seconda del gruppo di appartenenza. Un encoder tachimetrico è costituito da un anello con due o più tacche separate, che invia un impulso ogni volta che il riferimento (un punto prefissato del rotore) passa da una tacca ad un’altra differente. Il segnale così generato sarà proporzionale al numero di tacche attraversate in un determinato intervallo di tempo, fornendo un’indicazione di velocità. Un encoder relativo invece restituisce direttamente un’informazione su velocità ed accelerazione del rotore, ma non sulla posizione. Un encoder assoluto, al contrario, fornisce l’angolo istantaneo di posizione del rotore. In quest’ultimo si utilizzano degli anelli concentrici presenti all’interno della carcassa, che allineandosi secondo varie combinazioni generano un segnale elettrico in uscita contenente le informazioni sull’angolo.

ALMATracker prevede l’utilizzo di due encoder rotativi assoluti ***ROTACOD HS58M*** posizionati solidali con i due assi di rotazione. Questi dispositivi sono in grado di misurare angoli con una precisione di 16 bit, che corrispondono a 2^{16}=65536

posizioni equidistanti. Volendo valutare la precisione in termini più comuni, si ottiene:

$$\alpha_{lim} = \frac{360\ deg}{65536} \cong 0.005493deg$$

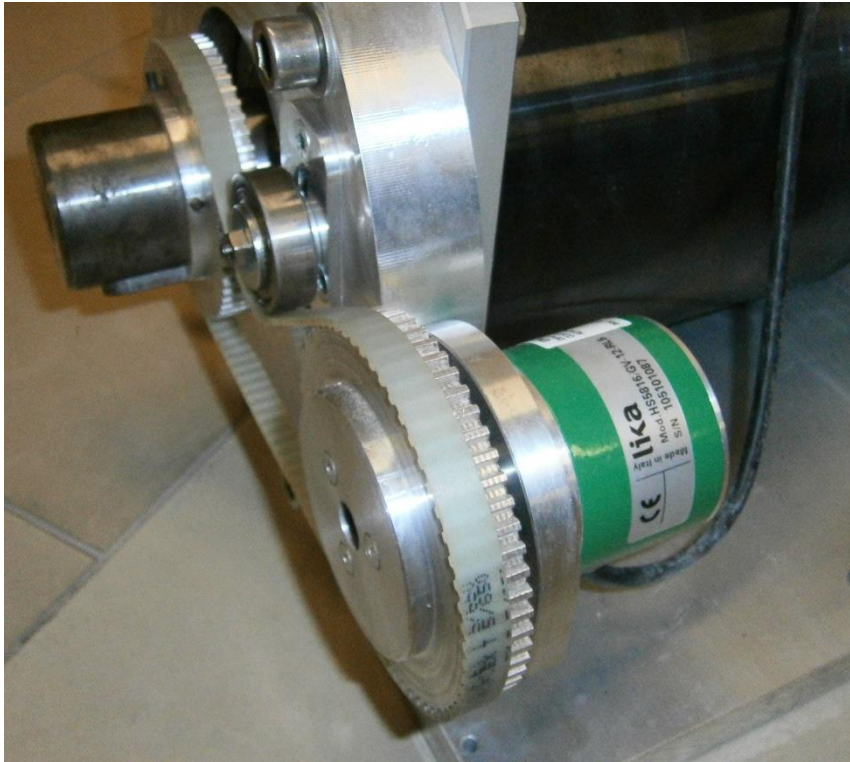



Figura 3 – Uno dei due encoder ROTACOD HS58M utilizzati per la lettura dei dati angolari sul sistema ALMATracker.

2.3.1. Interfaccia SSI

L'Interfaccia SSI (**Synchronous Serial Interface**) è un sistema di comunicazione dati "compatto" nato appositamente per ridurre il numero di connessioni necessarie tra encoder e sistema di elaborazione. Ad esempio encoder come quelli utilizzati per ALMATracker utilizzano un protocollo formato da 32 bit dove i primi 6 bit meno significativi rappresentano il controllo a ridondanza ciclica (CRC, Cyclic Redundancy Check), seguono due bit – uno di avviso e uno di errore – e concludono i rimanenti 16 bit che rappresentano l'angolo misurato espresso in codifica Gray. Il codice Gray assicura un minore rischio di errori nella misura dell'angolo poiché tra due angoli adiacenti la stringa varia soltanto di un bit. L'interfaccia SSI consiste nell'invio seriale di sequenze di bit al cui interno sono presenti informazioni di

posizione e stato dell’encoder. La sequenza di bit contenente la posizione angolare è contenuta all’interno di uno *shift register* dell’encoder. La lettura di tale registro avviene in maniera sincrona con una linea di clock di frequenza compresa fra 100 kHz e 10 MHz. E’ necessario inviare un treno di impulsi (tanti quanti sono il numero dei bit dello shift register), al quale l’encoder risponde restituendo per ogni fronte di discesa del segnale di clock un bit in uscita fino allo svuotamento dell’intero registro.

L’encoder è stato collegato con la cRio tramite un bus seriale standard RS422 differenziale. L’alimentazione dell’encoder è stata fornita con un alimentatore stabilizzato a 12 V DC. Tuttavia il modulo (NI9401) utilizzato gestisce segnali in ingresso ed uscita di tensione compresa nell’intervallo 0 – 5 V. Per questo si è reso necessario l’utilizzo di un circuito che realizzasse la traslazione di livello da 0 – 12 V a 0 – 5 V.

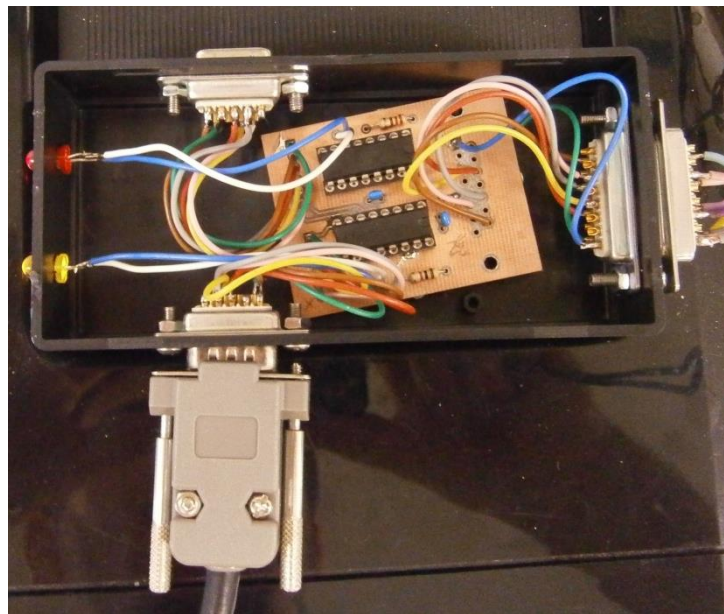



Figura 4 – Circuito per la traslazione dei livelli logici tra encoder e modulo NI9401.

2.4. *AZIONAMENTI*

Gli azionamenti sono costituiti da due motori brushless AC e due moduli di controllo. Motori e moduli di controllo sono connessi tramite bus dati proprietari, per il controllo, e da linee di potenza. Il collegamento tra gli azionamenti avviene tramite BUS RS485. Tramite una specifica interfaccia RS232/RS485 che si affaccia sul BUS, la cRIO si collega con la propria seriale RS232 al BUS RS485. Gli

azionamenti utilizzati nel progetto sono costituiti da un modulo di potenza ***Siemens Synamics Power Module 340,*** che alimenta il sistema, e dalla CPU ***Siemens CU305 DP***. I motori scelti per questo progetto sono dei ***Siemens 1FK7032*** dotati di encoder interno.

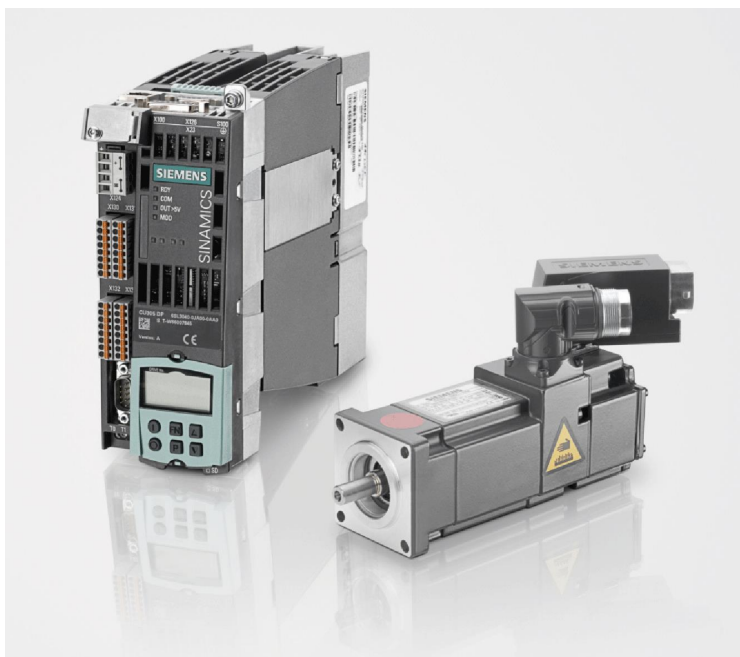



Figura 5 – Azionamento SINAMICS (a sinistra) e motore brushless (a destra). L’azionamento è necessario per la gestione del motore, e permette la comunicazione tramite protocollo Siemens USS®. Inoltre è dotato di un limitatore di velocità che permette di lavorare in sicurezza senza danneggiare il motore. Accedendo all’azionamento tramite porta di programmazione è possibile impostare molti dei parametri di controllo. All’interno del motore è presente un encoder che fornisce una misura sull’albero della velocità di rotazione.

2.4.1. Protocollo USS®

Il protocollo di trasmissione degli azionamenti Siemens, denominato USS®[1] (Universal Serial Interface protocol) è un protocollo di comunicazione seriale tra azionamento e macchina, che nel caso in questione è la cRIO. La connessione identifica prima di tutto un master e uno slave: come master si intende il dispositivo che comanda, cioè la cRIO; per slave il dispositivo comandato, cioè l’azionamento.

Il protocollo USS funziona nel seguente modo:

---

[1] USS® Protocol è un marchio registrato da Siemens.

1. Il dispositivo MASTER (cRIO) invia una stringa di controllo (*Task telegram*) allo SLAVE;
2. Il dispositivo SLAVE (azionamento) invia una stringa di risposta (*Response telegram*) al MASTER.

Da evidenziare che nella risposta, cioè nel *Response Telegram*, sono presenti due serie di dati dove i primi 8 byte rappresentano il dato vero e proprio di informazioni del motore, i secondi 8 byte sono un eco dell'ultima stringa di controllo ricevuta. Come si vedrà in seguito in entrambe le comunicazioni è presente un byte di controllo che indentifica il dispositivo cui si riferisce (ADR). Il protocollo USS può controllare fino ad un massimo di 31 azionamenti (ADR da 0 a 30) ai quali invia i rispettivi Task Telegram, ricevendone la risposta.

La struttura del telegramma, inviato e ricevuto, presenta una parte comune. Il telegramma è diviso in pacchetti che, nel caso di ALMATracker, sono quelli mostrati in **Figura 6**.

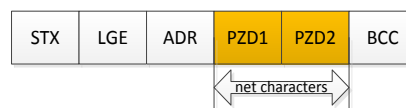



Figura 6 – Struttura dati del protocollo USS®.

COD	Denominazione	Lunghezza	Contenuto
STX	Start of Text		Carattere ASCII (02 HEX)
LGE	Telegram Length	1 byte	Lunghezza del carattere (ADR+PZD+BCC) espressa in numero binario
ADR	Address Byte	1 byte	Indirizzo dello SLAVE, in codifica binaria
PZD	Net Characters	2 byte/char	Le informazioni dipendono dalla direzione di trasmissione
BCC		1 byte	Carattere di controllo

Tabella 1 – Tabella dei pacchetti di cui sono composti i telegrammi secondo il protocollo USS®.

Nel *Task telegram* PZD1 prende il nome di *Control word* e contiene 16 *flags*[2] necessari per l'accensione e il controllo del dispositivo comandato. Il byte successivo PZD2 prende il nome di *Main setpoint* e contiene il set di velocità che si vuole impostare all'azionamento specificato nel campo ADR.

Nel *Response telegram* PZD1 è denominato *Status word* e contiene 16 *flags* che riportano lo stato del motore. PZD2 prende il nome di *Main actual value* e contiene il set di velocità letta dall'encoder interno al motore dell'azionamento specificato nel campo ADR.

2.5. MECCANICA

La movimentazione dell'antenna parabolica dell'ALMATracker è affidata ad un sistema motorizzato a due assi. Il sistema è composto da due assi motorizzati identici montati a novanta gradi uno su l'altro costituiti da una base rettangolare in alluminio (660 mm x 320 mm x 20 mm) su cui sono ancorati tutti gli elementi. Su ognuna delle basi sono stati posizionati i motori Siemens. Il primo riduttore è calettato direttamente sul motore. Tramite una trasmissione a cinghia l'uscita del primo riduttore va a movimentare la seconda coppia di riduttori del sistema. E' stato scelto un sistema a doppio riduttore per poter riuscire a recuperare i giochi dovuti agli ingranaggi. I due riduttori hanno rispettivamente rapporti di demoltiplica pari a 28:1 e 70:1. Ciò permette di rallentare la rotazione ad elevata velocità e bassa coppia del motore ed ottenere bassa velocità ma elevata coppia[3]. Questa scelta è giustificata se si tiene conto che la parabola utilizzata dal sistema ha un diametro di circa 3 m, realizzata in lega leggera di alluminio, per una massa di circa 80 kg. Questo fatto introduce un elevato momento di inerzia che richiede un'elevata coppia.

---

[2] Per la codifica esatta dei vari caratteri si rimanda alle specifiche del protocollo Siemens USS® disponibile su http://cache.automation.siemens.com/dnl/DU0MjczAAAA_24178253_HB/uss_24178253_spec_76.pdf

[3] La velocità dell'asse finale può dunque essere calcolata come $\omega_{asse\ finale} = \omega_{motore}/(28 * 70)$.

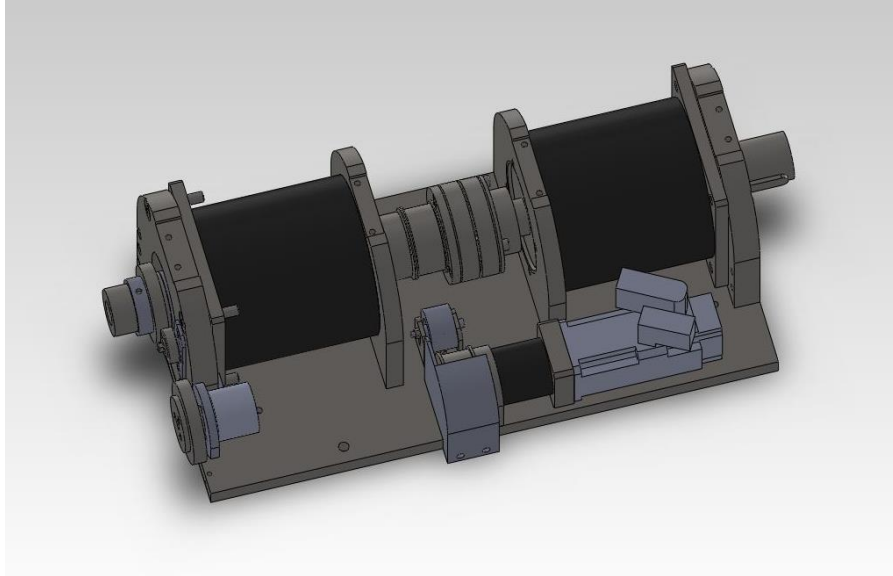



Figura 7 – Rendering Solidworks del singolo asse del sistema ALMATracker. Sebbene in figura non siano mostrate le cinghie, si possono notare le varie pulegge su cui le cinghie stesse andranno alloggiate. Il rettangolo grigio chiaro rappresenta il motore brushless, al quale è collegato il primo riduttore (28). Tramite cinghia il moto è poi trasmesso ai due riduttori maggiori (70) dai quali escono gli assi finali. Nell'asse uscente dal riduttore a sinistra è presente un secondo cinematismo a cinghia, sulla cui ruota condotta è montato l'encoder.

La struttura finale dell'ALMATracker sarà composta da due assi identici, dove il primo sarà ancorato al suolo e movimenterà il secondo asse più parabola. Il secondo posizionato sopra il primo andrà a movimentare direttamente la parabola in direzione ortogonale al primo (**Figura 8**).

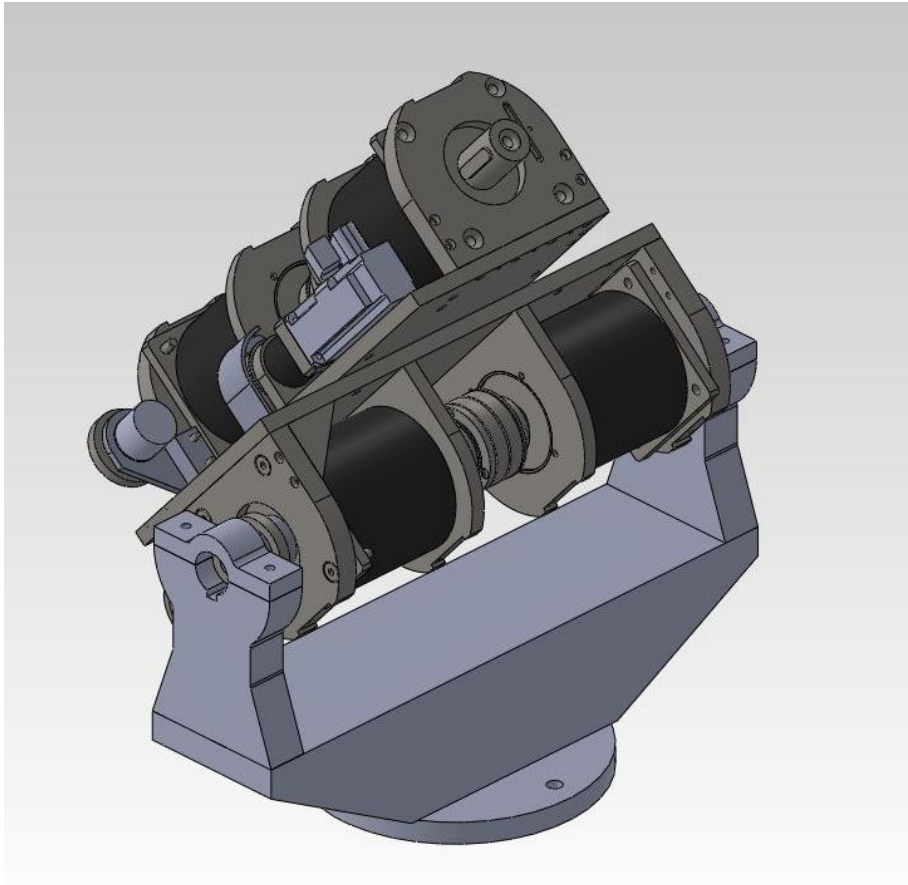



Figura 8 – Rendering Solidworks del sistema di assi di ALMATracker.

La parabola ha dunque due gradi di libertà indentificati nei due angoli X e Y che individuano univocamente la direzione di tracking. Questi due angoli sono tuttavia limitati dalla meccanica del sistema, ed in particolare occorre tenere conto dei punti in cui le basi degli assi giungono a contatto con i supporti. Considerando quindi le possibili collisioni, otteniamo una limitazione dell'operatività dei due assi tra un angolo minimo ed uno massimo, verosimilmente diversi per i due assi. Per tenere conto della limitazione dell'escursione massima sono stati previsti due microinterruttori per ogni lato di ogni asse (cioè ridondanti due per l'angolo massimo e due per il minimo). Gli interruttori vengono fissati alla base dell'asse tramite una staffa in alluminio. Questi interruttori hanno lo scopo di fornire al software un segnale di avviso di collisione imminente, e sono attivati dal passaggio della lastra ad un certo angolo.

Capitolo Terzo

ARCHITETTURA SOFTWARE

Il software di controllo e gestione di ALMATracker si sviluppa su tre livelli differenti: FPGA, cRIO e PC. In particolare l'algoritmo deve essere racchiuso in catena di comando, cioè ogni dispositivo deve utilizzare valori impostati o feedback provenienti dagli altri.

Lo snodo centrale del sistema è l'algoritmo di controllo e movimentazione che deve essere implementato sul sistema centrale (cRIO) e deve svolgersi in ciclo chiuso con gli altri algoritmi. Uno schema approssimativo del sistema può essere osservato in **Figura 9**.

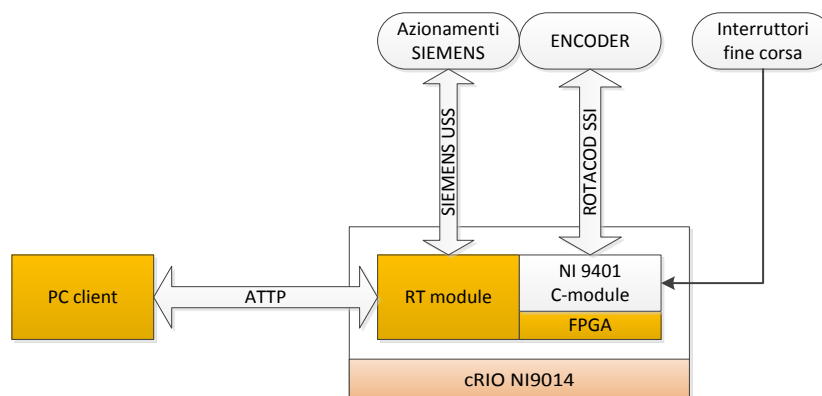



Figura 9 – Architettura del sistema ALMATracker in cui si evidenziano le comunicazioni tra i vari componenti hardware.

Il modulo FPGA ha come scopo l'acquisizione di dati dall'encoder e dei segnali di fine corsa. Inoltre si dovranno generare i valori di conteggio per il successivo calcolo delle velocità. Sempre all'interno della cRIO vi è il modulo RT che ospita la VI più complessa poiché deve svolgere molteplici compiti elencati di seguito:

- comunicazione TCP/IP da/verso PC client;
- comunicazione da/verso azionamenti Siemens;
- elaborazione dei dati acquisiti da FPGA;

- elaborazione dei dati complessivi per il controllo e la movimentazione degli assi.

Il segmento PC deve acquisire tutti i comandi impostati dall'utente, gestendo opportunamente l'interfaccia grafica. Si deve inoltre provvedere alla comunicazione bidirezionale via TCP con la cRIO, dalla quale si acquisiscono i dati di sistema che devono essere visualizzati sul dispositivo.

3.1. INTERFACCIA FPGA

Il modulo FPGA si occupa della lettura dei dati dagli encoder, degli indicatori di fine corsa (si veda **capitolo 5.0.3.** Contatti di fine corsa) e di una prima elaborazione del dato angolare. Questo aspetto richiede l'accesso diretto ai segnali in ingresso dal C-module. La lettura delle posizioni degli encoder viene espressa come intero decimale per l'impossibilità di utilizzo della virgola mobile. Inoltre viene eseguito un conteggio degli impulsi e del periodo per il calcolo della velocità dell'encoder.

3.1.1. Acquisizione dato angolare

Come introdotto nel **capitolo 2.3.1.** Interfaccia SSI per ottenere i dati dall'encoder è necessario interrogare il dispositivo con un treno di 32 impulsi con logica negativa. Gli impulsi devono presentare un periodo di 2.4 μs e il periodo tra l'inizio di due treni consecutivi deve essere di 100 μs.

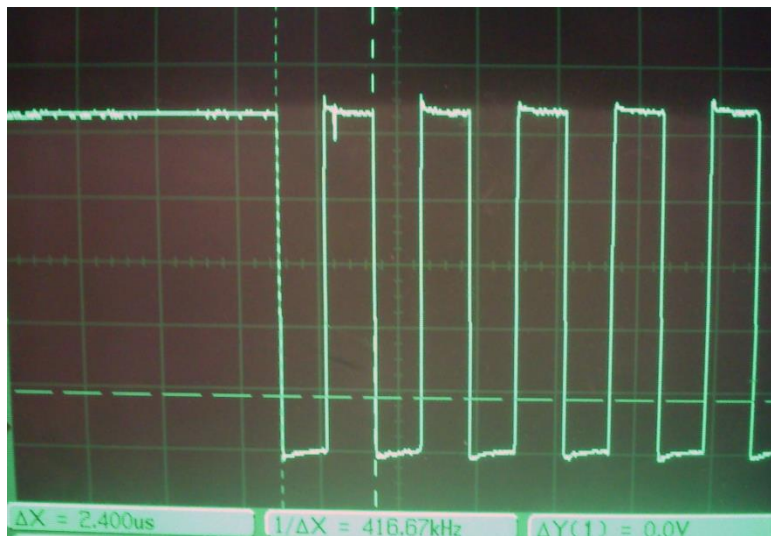



Figura 10 – Misurazione all'oscilloscopio di un singolo impulso, del quale si può notare il periodo pari a 2.4 μs.

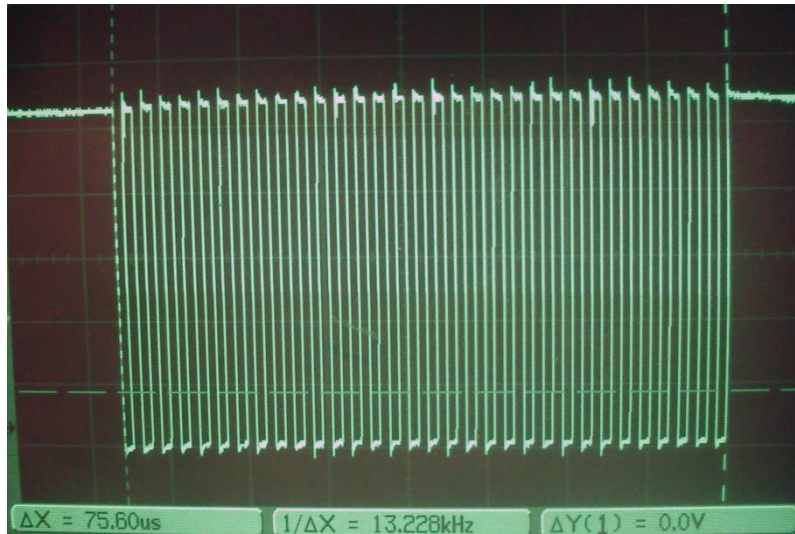



Figura 11 – Misurazione all'oscilloscopio del segnale in uscita dalla porta DIO0. In basso è possibile notare la durata del singolo treno di impulsi di 75.60 μs, identificata dai riferimenti tratteggiati verticali.

Per ottenere il dato si lavora nel modulo FPGA generando il treno di impulsi e leggendo il dato in uscita dall'encoder. All'interno di una struttura sequenziale si utilizza il primo frame per porre il segnale di output alto[4] sui canali utilizzati (DIO4 per l'asse X e DIO5 per l'asse Y). In un successivo frame si hanno tre cicli separati, uno per la raccolta del dato da encoder e due per il calcolo della velocità. Il ciclo che realizza la raccolta del dato è al suo interno diviso in un ulteriore struttura sequenziale. Il primo frame impone la cadenza dei treni di impulsi utilizzando un *Wait Until Next Multiple* settato ad un valore di 100 μs. Nel secondo frame un *ciclo For* esegue 32 iterazioni nelle quali, ancora una volta con una struttura sequenziale, manda il segnale basso nel primo frame (impostando il valore falso sul rispettivo DIO) e nel secondo attende 1 μs, lo riporta alto e attende un altro μs. La forma d'onda è così generata e per tutta la durata del secondo frame l'encoder mette a disposizione il dato sui canali DIO0 per l'asse X e DIO1 per l'asse Y, che vengono inseriti in due differenti array e memorizzati in uno *shift register*. In uscita dal *ciclo For* i due dati angolari sono disponibili all'interno di due array. Nei frame successivi si utilizza la funzione *Array Subset* per estrapolare dall'array i bit utili, cioè 16 bit a partire dal sesto e il dato viene convertito da codifica Gray ad intero decimale con un

[4] Il modulo NI 9401 ha un output di 5 V DC per un segnale alto e 0 V per uno basso.

semplice algoritmo. A questo punto il dato è stato ottenuto e sia encoder che sistema di controllo sono pronti ad un nuovo treno di impulsi e quindi allo scambio di un nuovo dato.

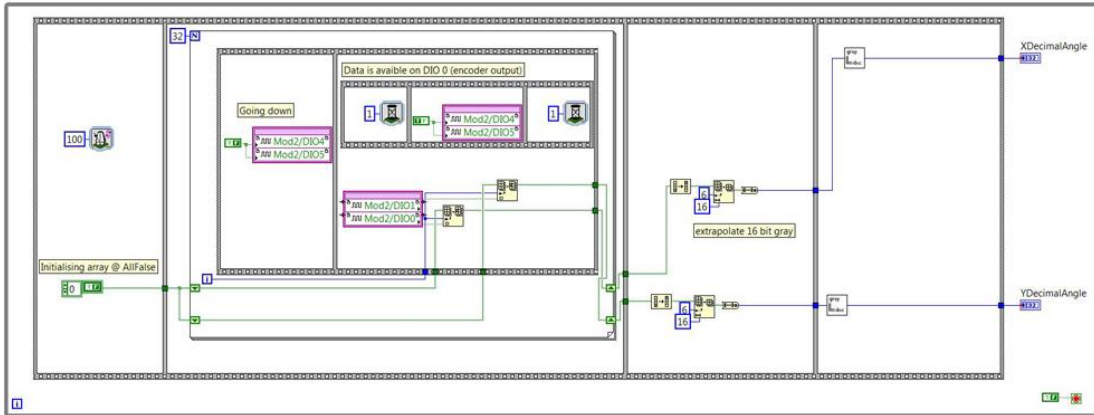



Figura 12 – Ciclo di generazione del treno di impulsi e lettura del dato da encoder. All'interno del modulo FPGA i blocchetti di colore viola indicano la comunicazione con i moduli inseriti nello chassis. In questo caso il modulo NI 9401 era stato inserito nel secondo posto, per cui è indicato dall'FPGA come "Mod2". L'informazione data da DIO (Digital Input/Output) si riferisce all'ingresso/uscita del modulo.

1:10 flags	11:26 angolo	28:32 controlli

Figura 13 – Struttura del dato letto da encoder sotto forma di array di boolean. Per ottenere l'angolo è sufficiente utilizzare la funzione *Delete from Array* per eliminare i bit in eccesso, conoscendone la posizione all'interno del registro.

La funzione che si occupa della trasformazione del dato angolare da binario Gray ad intero decimale è racchiusa nella subVI *sF_GrayToDecimal.vi*. Questa funzione utilizza un *ciclo While* in cui due *shift register* vengono inizializzati l'uno al valore dell'angolo in Gray, uno a 0. Il primo registro esegue ad ogni iterazione lo spostamento dei bit verso sinistra (*Logical shift*) eliminando così il primo bit a sinistra per ogni iterazione. Il valore decimale dell'angolo è ottenuto per somma logica esclusiva (XOR) tra il valore uscente dal primo registro e quello del secondo. Il ciclo termina quando il valore residuo del registro è nullo, cioè la condizione "registro>0" risulta falsa. Il secondo *shift register* nel frattempo esegue ad ogni ciclo la somma modulo 2 tra i valori uscenti dai due registri. In uscita al ciclo il dato decimale è disponibile nel secondo registro.

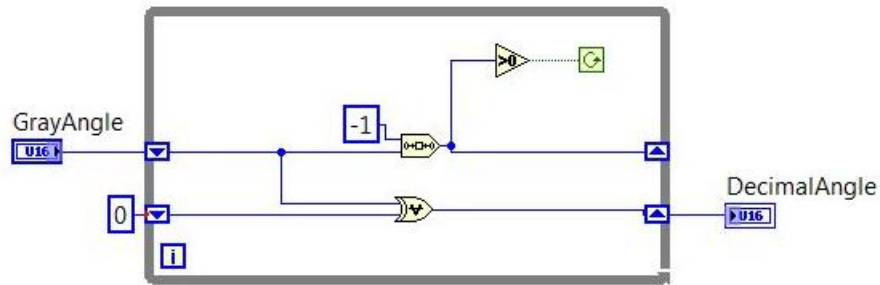



Figura 14 – *sF_GrayToDecimal.vi*

3.1.2. Generazione dei segnali di velocità

Se si esegue una valutazione sulla frequenza di cambiamento del dato dell'encoder occorre considerare una velocità massima operativa del motore di 3000 RPM per cui si ottiene una velocità massima all'asse finale di circa 1.5306 RPM. Poiché una rotazione completa dell'encoder fornisce 65536 variazioni, e la frequenza è pari al reciproco del periodo, la frequenza di variazioni a 3000 RPM sarà pari a

$$f_{vel\,max} = \left(\left(\frac{3000RPM}{28*70}\right)*65536\frac{step}{giro}*\frac{1\,h}{60\,sec}\right)^{-1} = 1.672\,KHz$$

Dal momento che la frequenza massima dell'FPGA è di 40 MHz, cioè di un *Tick* di sistema ogni 0.025 μs, è possibile utilizzare una frequenza di lettura dei dati di almeno tre ordini di grandezza superiore alla frequenza massima di aggiornamento del dato. Dal sistema di acquisizioni dei dati dall'encoder è noto che il tempo minimo di acquisizione è di 75.60 μs[5]: al di sotto di questo *timing* i treni di impulsi emessi sono troppo ravvicinati e l'encoder non risponde correttamente. Da qui deriva la scelta del tempo di temporizzazione del ciclo di acquisizione da encoder di 100 μs cioè di una frequenza di 10 KHz che è almeno tre ordini di grandezza superiore alla $f_{vel\ max}$ appena calcolata. In questo modo si è certi di avere che tra due letture consecutive il dato angolare cambia al più di un bit. Più precisamente vi saranno

[5] Verificato in laboratorio anche con l'utilizzo di un oscilloscopio sul treno di impulsi in uscita verso encoder (**Figura 10**).

alcuni cicli per cui il dato rimane costante e uno in cui si modifica e così via. E' allora possibile creare una funzione che elabori un segnale per il conteggio dei tick che intercorrono tra due successive variazioni del dato. Questo si ottiene ponendo su FPGA due cicli in parallelo al ciclo di acquisizione da encoder. In particolare si dovrà generare un segnale per ogni asse, chiamato *Pulse*, che generi un gradino unitario ogni volta che il dato angolare si modifica, corredato da un altro segnale, chiamato *Direction*, che detiene informazioni circa il verso della rotazione (*Direction* $\in [-1,0,1]$) definito come la funzione segno della variazione angolare. Contemporaneamente nel secondo ciclo si esegue il confronto tra il valore attuale ed il precedente di *Pulse* creando un segnale che risulta vero sui soli fronti di salita dell'impulso. La distanza tra i fronti di salita è cronometrata in termini di *tick* che intercorrono in un certo periodo di campionamento fornito dal segnale *Sample* secondo un ciclo temporizzato ogni numero *Count tick*. Ciò equivale a dire che la funzione realizzata conta quanti istanti sono passati tra successive variazioni del dato angolare, avendo effettuato la misura ogni volta che il segnale *Sample* è cambiato di stato e secondo una temporizzazione di misura pari a *Count tick*.

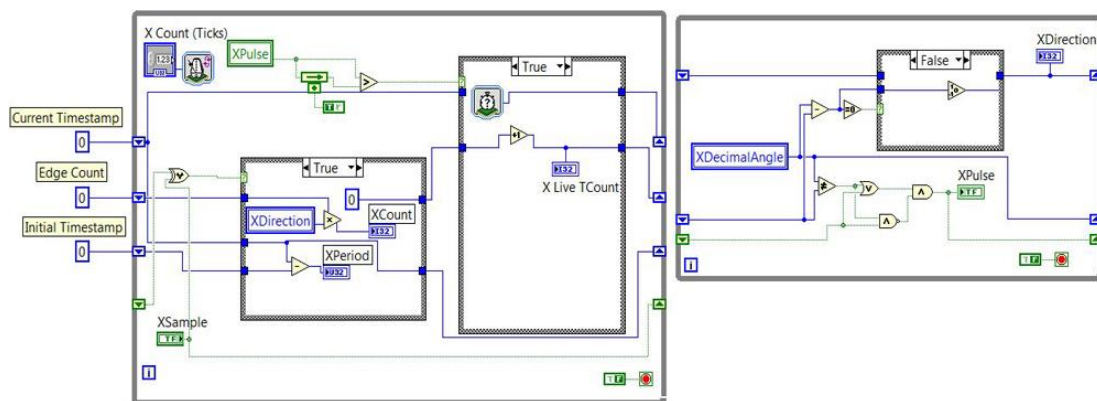



Figura 15 – Stralcio di codice della VI operante su FPGA. Il ciclo a destra serve alla generazione dell'onda quadra *XPulse* e della direzione di rotazione. Queste variabili sono poi utilizzate nel ciclo a sinistra per il conteggio dei *Tick* e del periodo.

La realizzazione di questo algoritmo richiede di completare una tabella di verità per il calcolo della funzione *Pulse*. In particolare, utilizzando la funzione si avvale di tre porte logiche – un OR, un AND ed un NAND. La tabella di verità del circuito realizzato, si riporta in **Tabella 2**.

A	B	1	2	3
0	0	0	0	1
1	0	1	1	1
0	1	1	1	1
1	1	1	0	0
0	0	0	1	0

Tabella 2 – Tabella di verità del circuito logico in cui si identificano: **A** segnale in uscita dall'operatore ≠; **B** segnale di uscita dallo *shift register*; **1** segnale di uscita dall'OR; **2** segnale di uscita dall'AND nonché segnale *Pulse* e nuovo valore dello *shift register*; **3** segnale di uscita dal NAND.

L'onda quadra *Pulse* contiene quindi l'indicazione di cambiamento dato all'interno dei fronti di salita e per questo viene filtrata da una comparazione tra un *Feedback Node* e la variabile stessa. Questa comparazione risulta vera solo nel fronte di salita, cioè quando 1 (segnale alto) > 0 (segnale basso) del ciclo precedente. Il valore del comparatore attiva un *case* che non modifica nulla se è falso (se l'angolo non è cambiato) mentre reinizializza lo *shift register* di periodo ed incrementa quello di conteggio tick nel caso di fronte di salita su *Pulse*. Un ulteriore case, attivato dal cambiamento di stato del segnale sample ottenuta con uno XOR tra dato corrente e dato precedente, fornisce il conteggio dei tick (con segno). L'attivazione di questo case fa si che si calcoli il *Count tick* ottenendone il valore da *shift register* (moltiplicato per il valore *Direction*), mentre il *Period* è ottenuto per differenza tra il tempo dell'ultima modifica dell'angolo e quello dell'ultima attivazione del case descritto.

3.1.3. Generazione dei segnali di fine corsa

Le due porte in ingresso rimaste libere – DIO2 e DIO3 – sono state utilizzate per i sensori di fine corsa dell'asse X. Questi sensori vengono realizzati con degli microinterruttori che vengono attivati fisicamente per contatto quando l'asse oltrepassa un certo angolo limite nelle due direzioni di rotazione. L'attivazione dell'interruttore cambia lo stato delle variabili *X max/min collision*, ed avrà su cRIO un effetto di limitazione sull'escursione angolare effettuabile. Dal momento che nella versione finale del sistema saranno presenti questi microinterruttori su entrambi gli assi, questi due segnali vanno ad attivare anche le corrispondenti variabili dell'asse

Y. Questo non ha alcun senso teorico, ma è utilizzato al fine pratico per testare la limitazione del range di entrambi gli assi, non disponendo di sufficienti porte in ingresso (necessarie due per asse). Quindi per poter implementare un vero controllo sulla versione finale di ALMATracker sono necessarie almeno quattro porte in ingresso oltra a quelle utilizzate per la comunicazione con l'encoder. Per ragioni di sicurezza sarà obbligatorio l'utilizzo di due moduli digitali I/O per avere almeno 16 canali così da poter avere per ogni angolo limite almeno due microinterruttori ridondanti che possono poi essere collegati via software tramite porta OR, presentando un'affidabilità superiore in caso di rottura del singolo componente.

3.2. INTERFACCIA CRIO

Come già detto la VI operante sul modulo RT della cRIO è lo snodo centrale del software. In particolare si devono svolgere contemporaneamente azioni di trasmissione dati da e per Client, azionamenti ed FPGA. Tutto ciò richiede una quantità di cicli separati che operino in parallelo, tutti temporizzati secondo un timing adeguato al loro scopo ma comunque mai inferiore ai 10 ms per evitare il sovraccarico della capacità di calcolo disponibile. Ogni ciclo esegue una funzione diversa, riportata di seguito:

- Comunicazione con FPGA;
- Impacchettamento dati;
- Ricezione/Trasmissione via TCP/IP;
- Controllo e movimentazione;
- Comunicazione con azionamenti Siemens.

Il primo punto racchiude tutte quelle operazioni che riguardano la ricezione dei dati dell'encoder dal modulo FPGA e la loro elaborazione fino ad ottenere i dati di posizione e velocità. Nei cicli di impacchettamento dei dati e trasmissione TCP si ha la preparazione dei dati in funzione della frequenza di aggiornamento richiesta e l'inserimento in coda della richiesta di spedizione. Parallelamente, si provvede alla loro spedizione, oltre che al ricevimento dei dati di comando dal client come verrà approfondito nel capitolo quarto. I comandi derivanti dalla lettura TCP insieme ai dati derivanti da FPGA vengono poi portati nel ciclo di movimentazione per essere

elaborati fino ad ottenere un setpoint di velocità. Tale setpoint viene poi importato nel ciclo di trasmissione con gli azionamenti che provvede ad inviarlo secondo il protocollo USS tramite l'algoritmo studiato nel capitolo 3.2.3.

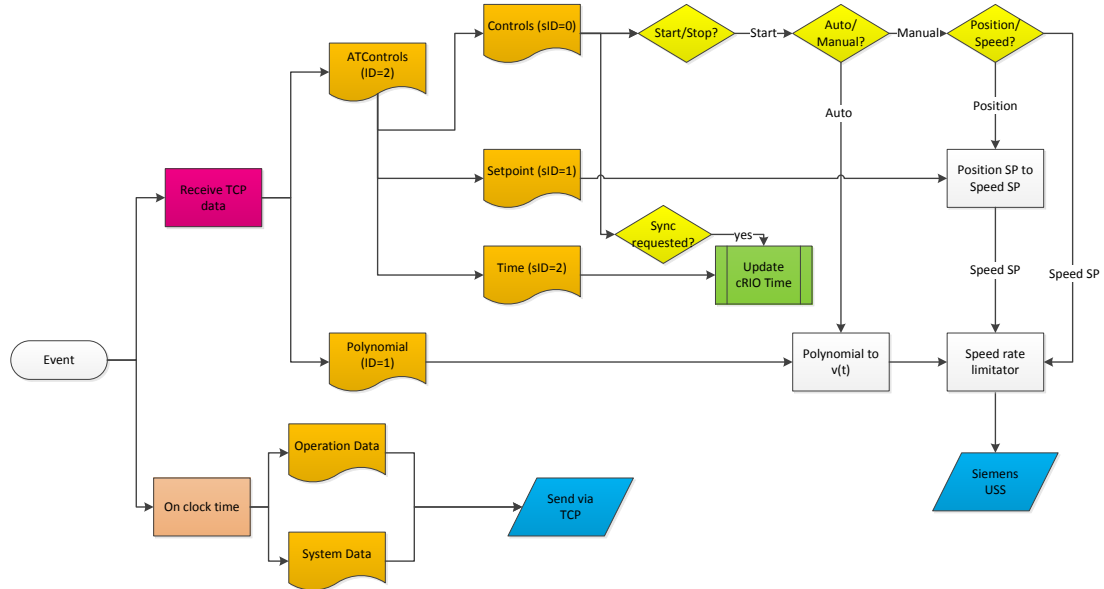



Figura 16 – Schema logico del software contenuto su cRIO. La complessità progettuale è ben visibile nella moltitudine di connessioni, ma viene districata in fase di stesura del programma grazie all'utilizzo di variabili locali che mettono a disposizione un dato continuamente aggiornato a più utenze. Nello schema qui sopra è stata volontariamente tralasciata la parte di elaborazione del dato dell'encoder (per evitare di complicare il grafico), che è a disposizione di tutti i blocchetti bianchi in cui è necessario il feedback.

3.2.1. Elaborazione dati encoder

Per l'interpretazione dei dati dell'encoder è necessario lavorare prima da lato FPGA, per quanto precedentemente descritto, fino ad ottenere l'angolo espresso come intero decimale. Il valore così ottenuto riporta il valore della posizione espresso in un sistema a 16 bit dove l'intero angolo di 360° è rappresentato da un valore pari a 2^{16}=65536. Per ottenere il valore in gradi è necessario un calcolo in virgola mobile, da eseguire dunque su modulo RT. L'intero decimale (*DecimalAngle*) viene convertito in angolo espresso in gradi (*Angle*) tramite la semplice relazione:

$$Angle = \frac{DecimalAngle * 360°}{65536}$$

Questo calcolo viene effettuato nella subVI *sR_DecimalToDegree.vi*.

3.2.2. Calcolo delle velocità

I dati provenienti dagli encoder individuano soltanto la posizione angolare dell'antenna. Tuttavia, avendo un sistema comandato in velocità, è necessario ottenere in qualche modo anche una stima della velocità angolare istantanea (ω). Ricordando che:

$$\omega = \frac{d\alpha}{dt}$$

il modo più intuitivo di realizzarlo sarebbe quello di calcolare in un determinato intervallo di tempo (dt) la variazione di angolo (dα), ed ottenere la velocità semplicemente dividendo l'angolo per il tempo. Tuttavia con tale metodo si incorre in problemi di sottostima della velocità angolare nel caso $\omega_{effettiva} < \omega_{limite}$. Per la valutazione di ω_{lim} si può considerare $\alpha_{lim} \cong 0.005493$ deg (**capitolo 2.3.** Encoder) e un intervallo di campionamento dt=1 ms[6], si ottiene così:

$$\omega_{lim} = \frac{d\alpha_{lim}}{dt} = \frac{0.005493\ deg}{0.0001\ s} \cong 5.493\ deg/sec$$

In base a questo risultato, dunque, non è possibile rilevare in tal modo velocità inferiori a ω_{lim} se l'intervallo di campionamento è di 1 ms, giungendo, nel peggiore dei casi, ad avere una velocità nulla per tutto il tracking se la velocità angolare è sempre minore di ω_{lim}. Si potrebbe dunque pensare di alzare l'intervallo di campionamento *dt* abbassando di conseguenza ω_{lim}. Così facendo, però, si perderebbe sensibilità in termini temporali, portando tutto il sistema ad aggiornarsi a frequenze più basse nonché si incorrerebbe in velocità medie per intervalli di tempo alti che si potrebbero discostare molto dalla velocità istantanea reale.

Una seconda possibile soluzione consiste nel considerare un campionamento non temporale, ma angolare: cioè non si calcola la velocità allo scadere di un intervallo di tempo fisso ma al verificarsi della condizione per cui l'angolo varia di uno step di encoder (1 di 65536 corrispondente a 0.005493 deg). Tramite *shift register* si tiene

[6] Un intervallo di acquisizione di 1 ms può essere considerato un buon tempo di scansione al fine di ottenere la velocità, ma è stato modificato per altri motivi spiegati di seguito.

traccia del tempo e alla prima variazione di angolo si calcola la velocità istantanea[7]. In tal modo si evita il fatto di poter ottenere velocità nulle sempre, ma si accetta la possibilità di ottenere velocità medie nel caso in cui il sistema si muova a velocità inferiori all'angolo limite.

La soluzione adottata è tuttavia un'altra. Anziché lavorare nel dominio spaziale si lavora nel dominio temporale, sfruttando l'elevata frequenza di *Tick* del modulo FPGA. I dati così ottenuti da FPGA – cioè *Count* e *Period* – vengono letti dal modulo RT che ne esegue il calcolo finale di velocità, tenendo conto che un periodo pari a 40˙000˙000 *tick* corrisponde a 1 sec. In parallelo al ciclo principale viene aperto il canale di comunicazione con il modulo FPGA tramite l'istruzione *OpenFPGAReference.* In seguito si accede ad un *ciclo While* temporizzato a 100 ms, che rappresenta un timing adeguato alla frequenza di aggiornamento dei dati. In questo ciclo è presente una struttura sequenziale a due frame in cui si ha un primo frame di attesa di 1 ms per permettere ad un altro ciclo parallelo di aggiornare i valori di *Sample* e *Tick Count*. Nel secondo frame il blocco di comunicazione FPGA *Read/Write Control* permette di esportare nel modulo FPGA i valori di *Sample* e *Tick Count* e di leggere i dati presenti sugli indicatori dell'interfaccia del modulo FPGA, in particolare vengono importati i valori di *DecimalAngle, Count, Period* e gli indicatori di fine corsa hardware riferiti ai due assi.

---

[7] A causa dell'architettura logica del modulo FPGA (operazioni a virgola fissa) è necessario che il calcolo della velocità sia effettuato nel modulo RT con i dati di $\Delta\alpha$ e Δt provenienti da FPGA.

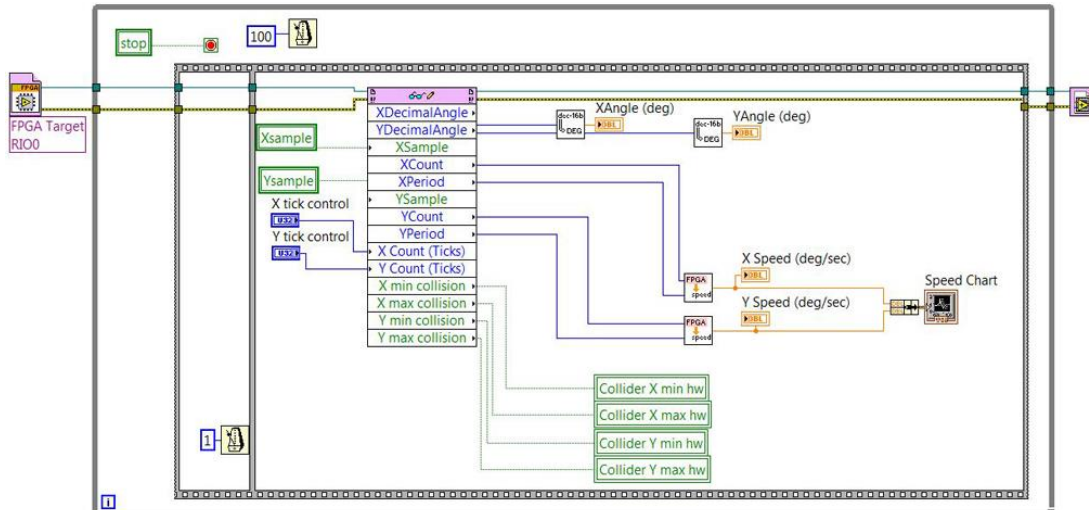



Figura 17 – Ciclo di acquisizione dei dati dal modulo FPGA. I blocchi a sfondo rosa sono riferiti alla comunicazione con FPGA.

Il valore dell'angolo in gradi trecentosessagesimali è ottenuto tramite una subVI che esegue il calcolo

$$Angle(\mathrm{deg}) = \frac{DecimalAngle * 360}{65536}$$

Il calcolo della velocità avviene nella subVI *sR_SpeedFromFPGA.vi*. In questo blocco si calcola la durata di acquisizione dei dati come il rapporto tra *Period* e la frequenza di acquisizione del modulo FPGA (40 MHz). Dividendo poi la durata di acquisizione per il numero di *tick* letti, contenuto nella variabile *Count*, si ottiene il valore del tempo trascorso tra due successive variazioni dell'angolo. Poiché la velocità è definita come il rapporto tra la variazione d'angolo ed il tempo ($\omega = \frac{\Delta\alpha}{\Delta t}$) ed essendo noto che la variazione di angolo è pari a

$$\Delta\alpha = \frac{720}{65536}$$

è possibile calcolare la velocità di rotazione dell'asse finale espressa in deg/sec. In questo calcolo occorre tenere presente che l'impulso *Pulse* viene analizzato sui soli fronti di salita, dunque una volta ogni due variazioni di angolo. Per questo l'angolo misurato attraverso il conteggio sarà pari a metà di quello reale. Questo coincide con il considerare, nel rapporto, che 65536 impulsi corrispondono al doppio di un angolo giro, ovvero 720 deg. Considerando inoltre il caso di velocità nulla in cui si avrebbe

una divisione con 0 a denominatore, per evitare errori di calcolo di velocità (cioè NaN, Not a Number), è stato inserito un *Select?* che imposta la velocità a 0 se *Count* è nullo o utilizza il valore calcolato altrimenti.

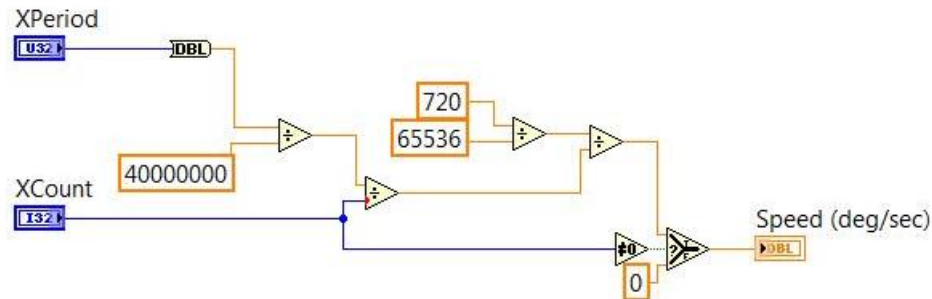



Figura 18 – *sR_SpeedFromFPGA.vi*

Il segnale *Sample*, così come *Count tick*, è spedito all'FPGA dal modulo RT secondo dei parametri che questa possiede. Il segnale di attivazione *Sample*, viene generato su RT come onda quadra unitaria di periodo fissato dal software in base alla velocità impostata ai motori. In particolare occorre variare il tempo di campionamento e il numero di *tick* da leggere, che dovranno essere maggiori tanto minore è la velocità del setpoint. Secondo le prove svolte in laboratorio si è giunti ad una tabella che riporta valori di velocità piuttosto soddisfacenti.

ω_{min}	ω_{max}	Sample time [ms]	Tick count
0	**0.1**	1500	100000
0.1	**1**	500	10000
1	**5**	200	10000
5	**10**	100	1000

Tabella 3 – Valori preimpostati sul modulo RT per il conteggio dei tick e del periodo. Per il confronto con la tabella occorre considerare la velocità in valore assoluto, espressa in deg/sec. Il confronto da effettuare comprende l'estremo superiore, ma non quello inferiore, cioè $\omega_{min} < |\omega| \leq \omega_{max}$. Per il solo valore di velocità nulla è compreso anche l'estremo inferiore dell'intervallo.

L'implementazione dell'algoritmo di generazione dei segnali di cui sopra avviene tramite due *cicli For* temporizzati paralleli. Nel primo ciclo vengono estrapolati i valori di *Tick Count* e del *Sample Timing*. Prima di tutto la velocità comandata al motore viene convertita in velocità all'asse e se ne calcola il modulo per poterla

confrontare con i dati tabulati. Il confronto avviene tramite una serie di *Select?* attivati da comparatori tra la velocità e i range di velocità tabulati. I vari *Select?* selezionano a seconda del range di velocità i corrispondenti valori di *Tick Count* e del *Sample Timing*.

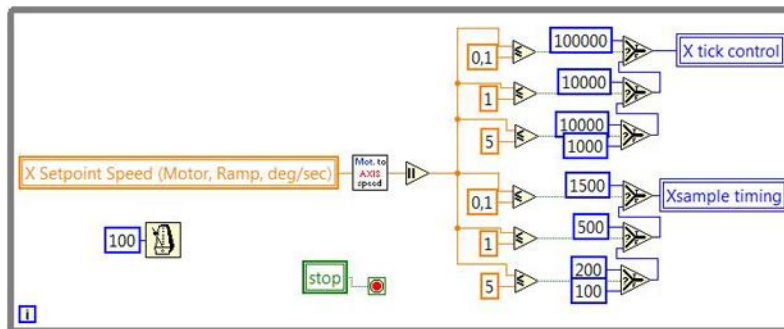



Figura 19 – Ciclo di selezione dei parametri *Tick Count* e *Sample Timing.*

Nel secondo *ciclo While* temporizzato secondo *Sample Timing*, si provvede alla generazione del segnale *Sample* tramite uno *shift register*. Il segnale infatti deriva dal valore stesso del registro, che viene cambiato ad ogni ciclo tramite una negazione logica.

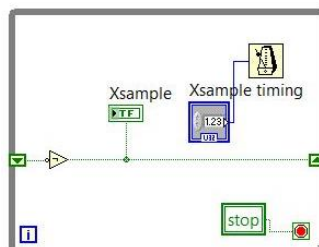



Figura 20 – Ciclo di generazione del segnale *Sample*.

Una serie di test effettuati in laboratorio è stata volta a mirare la validità dei valori tabulati per le basse velocità. Tramite una funzione si è creato un file composto da tre colonne contenenti le informazioni su un asse del setpoint di velocità impostata, del valore di velocità letta da encoder e di quella letta dall'encoder interno al motore. Il tutto è stato inserito in una serie di cicli annidati in modo tale che le misurazioni avvenissero:

- per diversi *tick count* (100000,10000,1000)
  - per diverse velocità (da 0 a 5,5 deg/sec con step di 0,5)

- per diversi *sample time* (da 100 ms con 14 step di 100ms)
    - per 50 campioni.

Il file è poi stato sottoposto a post processing utilizzando Matlab. L'algoritmo allegato in appendice "Almatracker report STS.m" ha fornito i seguenti risultati.

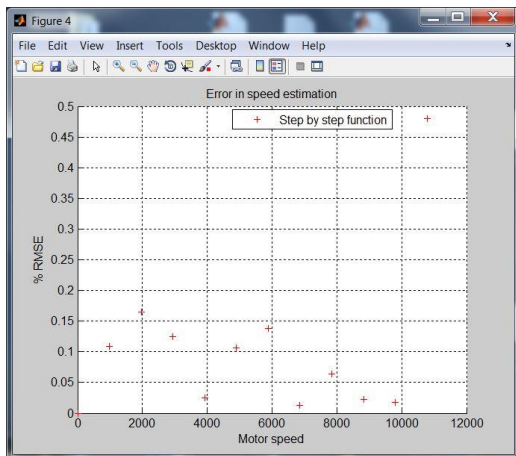



Figura 21 – Errore medio percentuale in relazione al setpoint di velocità del motore, utilizzando *Tick* e *Sample* prestabiliti per ogni velocità.

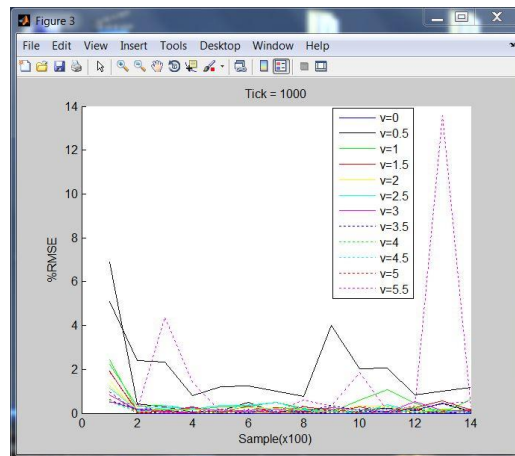



Figura 22 – Errore medio percentuale per le varie velocità alla variazione del *Sample* utilizzando un numero di *Tick* pari a 1000.

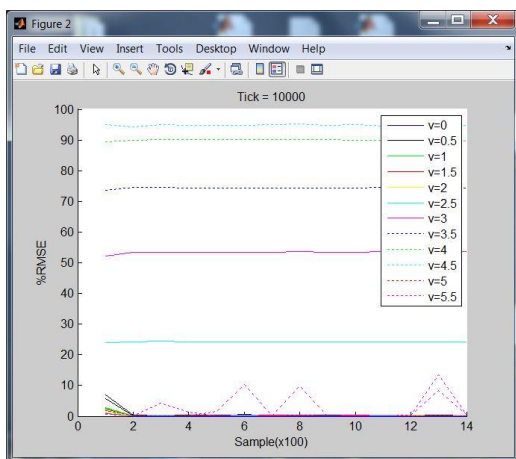



Figura 23 – Errore medio percentuale per le varie velocità alla variazione del *Sample* utilizzando un numero di *Tick* pari a 10000.

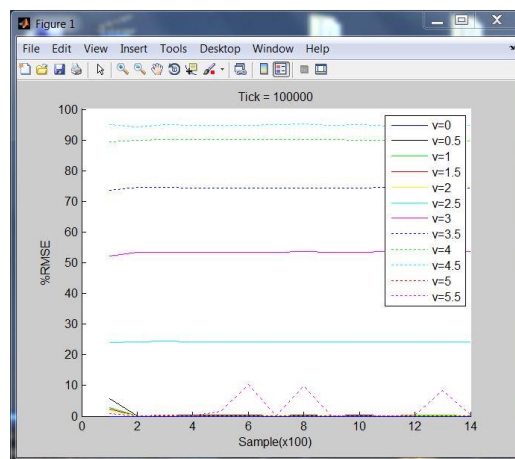



Figura 24 – Errore medio percentuale per le varie velocità alla variazione del *Sample* utilizzando un numero di *Tick* pari a 100000.

Il codice realizzato in Matlab ha analizzato i dati ottenuti da ALMATracker eseguendo una media dei 50 campioni di ogni misurazione e calcolando l'errore medio percentuale definito come:

$$RMSE\% = \frac{velocità_{SetPoint} - velocità_{Encoder}}{velocità_{SetPoint}} * 100$$

Nello screenshot di **Figura 21** si può osservare l'errore medio percentuale commesso utilizzando i dati tabulati in **Tabella 3**. Nello specifico si riportano di seguito anche i valori numerici ottenuti:

Velocità motore (deg/sec)	*Tick*	*Sample* (ms)	RMSE % riferito al setpoint
0	100000	1500	0.000000
980	10000	500	0.108959
2940	10000	200	0.124241
3920	10000	200	0.024637
4900	10000	200	0.106621
5880	10000	200	0.137907
6860	10000	200	0.012454
7840	10000	200	0.063998
8820	10000	200	0.023074
9800	10000	200	0.017976
10780	1000	100	0.479570

Tabella 4 – Calcolo dell'errore medio percentuale rispetto al setpoint impostato.

E' quindi intuitivo osservare che i valori di *Sample time* e *Tick count* scelti sono idonei, ottenendo un errore percentuale globale[8] sempre inferiore allo 0.5 %. Dai rimanenti plot di **Figura 22**, **Figura 23**, **Figura 24**, eseguiti per differenti *Tick count*, è inoltre evidente come sia avvenuta la scelta di questi valori. I valori sono stati scelti dopo alcune prove in cui si è notato che maggiore è la velocità più si richiede un *Tick count* basso. Tuttavia esiste un limite inferiore (1000 *Tick*) per ottenere una certa attenuazione del rumore. D'altro canto anche per *Sample* troppo bassi si ottiene una repentina variazione dei valori dell'errore che sono minimi per un *Sample* maggiore di 200 ms. Da notare inoltre che nel caso di basse velocità sono richiesti alti numeri

[8] comprensivo di rampe ed eventuali oscillazioni sul setpoint.

di *Tick* ma anche alti periodi di *Sample* per poter raccogliere un numero adeguato di campioni. Si evidenzia anche che per alte velocità, cioè oltre ai 5 deg/sec, il fatto di avere bassi *Sample* permette di avere un sistema più reattivo, cosa non possibile per basse velocità perché sarebbero soggette ad alti errori, come si evince dalla **Figura 24**.

3.2.3. Comandi motore (SiemensUSS.vi)

Per ottenere un software in grado di comunicare correttamente con gli azionamenti occorre rispettare rigidamente il protocollo sia in termini di incapsulamento dei dati sia in termini di timing. Occorre prima di tutto impostare alcuni parametri tramite il software di programmazione della Siemens connettendo il PC all'azionamento tramite la porta di programmazione. In questo caso è stato impostato un timeout di lettura di 5000 ms che impedisse che l'azionamento si bloccasse in assenza del dato in ricezione per il tempo prestabilito e l'indirizzo (ADR) dei motori a 0 per l'asse X e 1 per l'asse Y.

La subVI *sR_SiemensUSS.vi* si occupa della comunicazione bidirezionale con gli azionamenti, e viene richiamata nella VI principale all'interno di un *ciclo While* a parte non temporizzato. Questo non crea nessun problema in quanto tutto il codice contenuto nella subVI è già all'interno di un *ciclo While* temporizzato a 30 ms. Tale durata è stata accuratamente scelta come compromesso tra il tempo massimo di timout dei motori, che porterebbe alla loro disconnessione con generazione di errore, e la frequenza di aggiornamento dei dati in entrambe le direzioni tra modulo RT e azionamenti (circa 27 ms). Le uniche istruzioni della subVI che rimangono all'esterno del ciclo sono quelle inerenti alla configurazione della porta seriale. Tramite l'istruzione *VISA Configure Serial Port*, sono stati impostati i seguenti parametri:

- VISA resource name: ASRL1 (porta disponibile sulla cRIO);
- Baud Rate: 38400;
- Data bit: 8;
- Parity: Even;
- Stop bitss:1.0;
- Flow control: None;

- Enable terminator char: False;
- Timeout: 5000ms.

Una volta configurata la porta è stato impostato con il comando *VISA Set I/O Buffer Size* un buffer di dimensioni standard sui dati in spedizione e ricezione. Una volte effettuate queste operazioni si accede al *ciclo While principale*. Il ciclo è diviso in due parti, una di ricezione ed una di trasmissione. I valori in ingresso ed uscita alla subVI sono importati ed esportati con variabili globali, racchiuse nell'interfaccia *SiemensVariables.vi*.

Si comincia innanzitutto con la trasmissione dei pacchetti da MASTER a SLAVE, che richiede semplicemente di costruire il *Task telegram* impacchettando i dati di controllo in due byte (U8) contenuti nelle variabili PZD di Control word. I due PZD, chiamati PZD1 e PZD2, contengono il primo tutti i controlli per l'attivazione del motore, il secondo il setpoint di velocità richiesto. La parte di controllo, contenuta nel PZD1 è composta da 8 bit (cioè 8 comandi vero/falso) che devono essere usati in una determinata sequenza per l'accensione[9], la movimentazione e lo spegnimento dei motori. Gli 8 comandi distinti che devono essere opportunamente combinati secondo il protocollo USS sono:

1. Accensione;
2. Abilitazione operazioni;
3. Abilitazione rampa;
4. Abilitazione controllo rotazione;
5. Tacitazione anomalie.

Per motivi pratici le variabili di abilitazione rampa e abilitazione del controllo di posizione sono state poste a vero tramite una costante, istruzione lecita coerentemente al protocollo Siemens che non richiede un reale controllo su queste due variabili se non per usi particolari. Le variabili *Accensione* e *Abilitazione operazioni* sono attivate insieme per i due assi rispettivamente dalle variabili globali *On/Off* ed *Enable Operation*.

---

[9] Alla sequenza di accensione è dedicato l'intero paragrafo **3.2.4. Sequenza di accensione**.

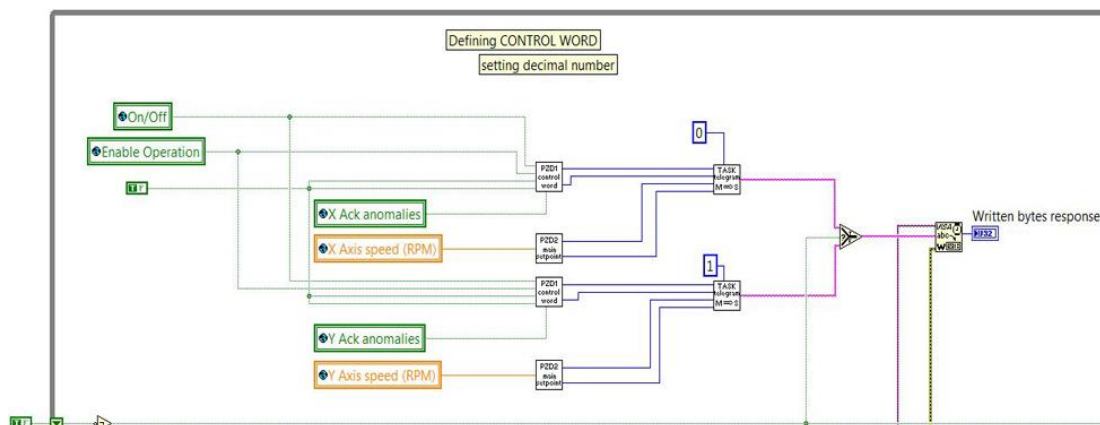



Figura 25 – Stralcio di codice rappresentante la parte di trasmissione di *SiemensUSS.vi* in cui si nota come il *Task telegram* sia ottenuto impacchettando i comandi motore ed il setpoint richiesto e venga subito inviato. Prima dell'invio un comando *Select?* decide quale pacchetto inviare a seconda del valore assunto dalla variabile booleana posta nello *shift register*. Questa variabile cambia ad ogni ciclo – grazie alla porta di negazione logica posta a fianco del registro – permettendo di inviare alternatamente i dati ai due diversi azionamenti.

La generazione dei due byte appartenenti a *Control Word* è demandata alla subVI *sR_PZDControlWord.vi*. I parametri booleani di ogni byte sono inseriti in un vettore tramite la funzione *Build Array*. I due byte del PZD sono infine convertiti in interi U8 con il comando *Boolean Array To Number*.

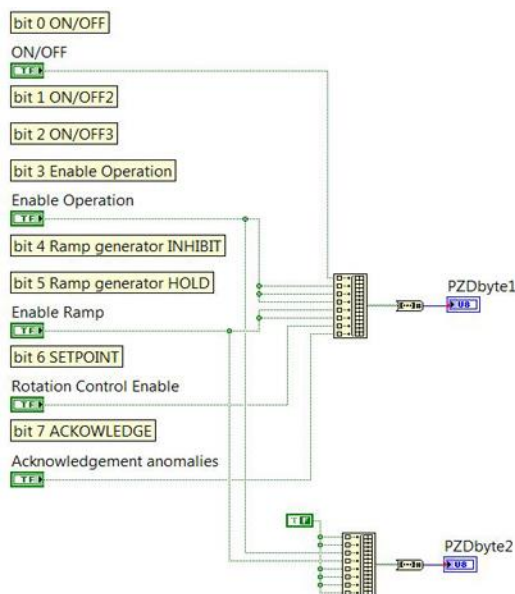



Figura 26 – *sR_PZDControlWord.vi* Lo scopo della VI è quello di creare i due byte che compongono il *Control Word* creando un vettore di parametri di azionamento, come specificato nel protocollo USS.

Il *Main setpoint* è ottenuto convertendo la velocità da double a due byte separati di interi (U8) nella subVI *sR_PZDMainSetpoint.vi*. La velocità da impostare

all’azionamento arriva qui sotto forma di double, espressa in RPM, dalla variabile *Axis Speed (RPM)*[10]. Per prima cosa si esegue una correzione di scala. Volendo riportare il valore di velocità (da protocollo comprese tra -6000 e +6000 RPM) attraverso un intero a 16 bit (valori compresi tra -32768 e +32767), mantenendo il massimo della precisione disponibile, occorre adattare la scala. Dunque occorre fare coincidere gli estremi, ed in particolare essendo una scala di valori simmetrica attorno allo 0, è necessario fare coincidere gli estremi minori. Riportando il valore di -6000 a -32768, si ottiene la proporzione:

$$RPM: I16 = -6000: -32768$$

da cui $I16 = RPM * \frac{32768}{6000} = RPM * 5.46133333333$. Successivamente si esegue la conversione in stringa tramite *Format Into String*, specificando il tipo di dato che si vuole ottenere tramite la stringa “%16b” che rappresenta i binari a 16 bit. Dal momento che l’azionamento richiede una stringa composta da un vettore di byte formattati come U8, si procede alla divisione della stringa tramite *String Subset* impostando 8 come numero di bit da leggere. Queste stringhe vengono infine convertite dal comando *Scan Value* secondo la codifica ad 8 bit (“%8b”) e riportati a due byte distinti in U8 tramite la funzione *To Unsigned Byte*.

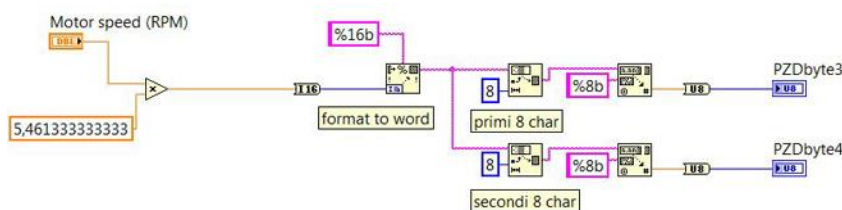



Figura 27 – *sR_PZDMainSetpoint.vi* Questa VI realizza la conversione completa da velocità ai due byte richiesti dal protocollo degli azionamenti.

I 4 byte così ottenuti vengono combinati in un array di byte U8 nell’ordine richiesto dall’azionamento, cioè:

- STX;
- LGE;

[10] La velocità espressa da questa variabile è riferita all’asse del motore.

- ADR;
- PZD1 byte2;
- PZD1 byte1;
- PZD2 byte1;
- PZD2 byte2;
- BCC.

Come si può notare i byte 1 e 2 del PZD1 devono essere invertiti per rispetto del protocollo USS.

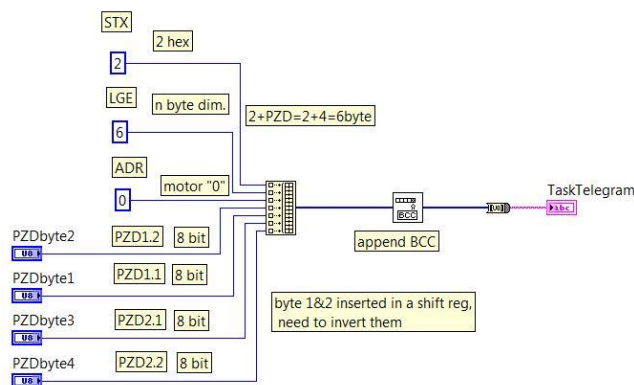



Figura 28 – *sR_TaskTelegram.vi* Assemblamento del *Task telegram* per il motore dell'asse X.

Ovviamente avendo due assi verranno assemblati due differenti *Task telegram*, che non possono tuttavia essere spediti contemporaneamente: si ricorre dunque ad uno *shift register*, che viene cambiato ogni ciclo di trasmissione, per inviare alternatamente i due telegrammi riferiti ai due diversi motori. La durata di un ciclo è di 30 ms, secondo la temporizzazione data con il comando *Wait Until Next Time Multiple*.

Passando alla ricezione si considera l'arrivo di un telegramma dall'azionamento.

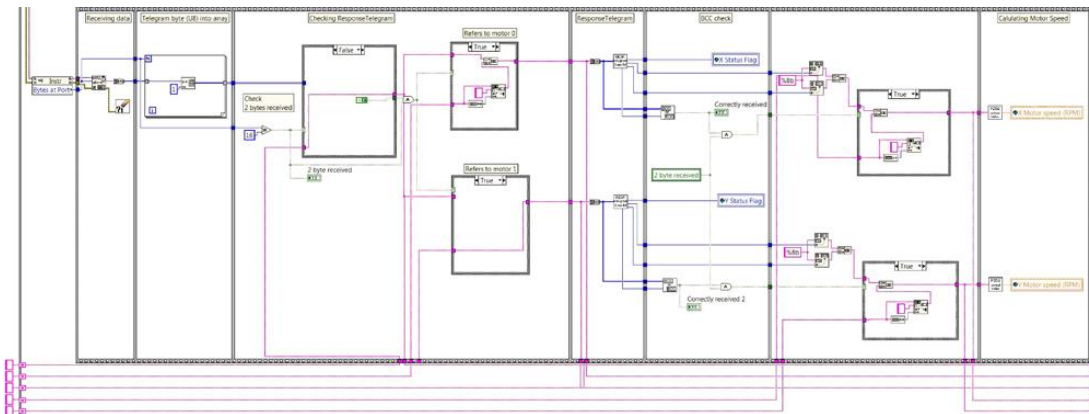



Figura 29 – Stralcio di codice rappresentante la parte di ricezione di *SiemensUSS.vi*.

Un *Property Node* impostato sull'opzione "*Number of Bytes at Serial Port*" permette di conoscere il numero di byte sono disponibili alla porta seriale impostata. Tale valore viene importato in una struttura sequenziale assieme al nome della porta. Nel primo frame il comando *VISA Read* legge i dati disponibili sulla seriale, composti dei byte specificati dal valore di cui sopra, inserendoli in una stringa. La stringa viene immediatamente convertita in un vettore di interi U8 tramite la funzione *String To Byte Array*. Nel frame successivo il vettore monodimensionale viene convertito in bidimensionale tramite un *ciclo For* di iterazioni pari ai byte ricevuti, creando un array tramite *Initialize Array*. Nel frame seguente si divide il dato ricevuto considerando l'azionamento al quale si riferisce. Un *case* esegue un primo controllo sul numero di byte letti. Se non tutti i byte sono ancora stati letti si accede al caso falso, in cui si usa una stringa memorizzata in uno *shift register* del ciclo esterno che viene impostata all'uscita del case senza altre operazioni. Nel caso vero, invece, il tunnel[11] richiede più passaggi. La stringa in uscita in questo caso risulta la concatenazione di due stringhe. La prima è quella derivante dalla subVI *sR_ByteToString.vi* collegando in ingresso il vettore 2D precedentemente calcolato. La seconda stringa deriva invece dal comando *Replace Substring* il cui ingresso è collegato al valore uscente dal registro e si occupa di sostituire con una stringa vuota i caratteri precedenti. Questo equivale a svuotare il registro se ci sono nuovi dati in arrivo, per evitare di incorrere in problemi di overflow delle stringhe. La subVI *sR_ByteToString.vi* calcola inoltre l'azionamento di provenienza del dato leggendo il

[11] In un *case*, per *tunnel* si intende il collegamento che c'è tra un ingresso ed una uscita.

valore corrispondente al terzo elemento della trasmissione (ADR). La lettura separata dei byte avviene dividendo il vettore con la funzione *Index Array*, in cui l'indice va da 0 a 7. Il vettore viene poi ricomposto tramite la funzione *Build Array*, e successivamente elaborato da *Byte Array To String* da cui si ottiene il dato sotto forma di stringa.

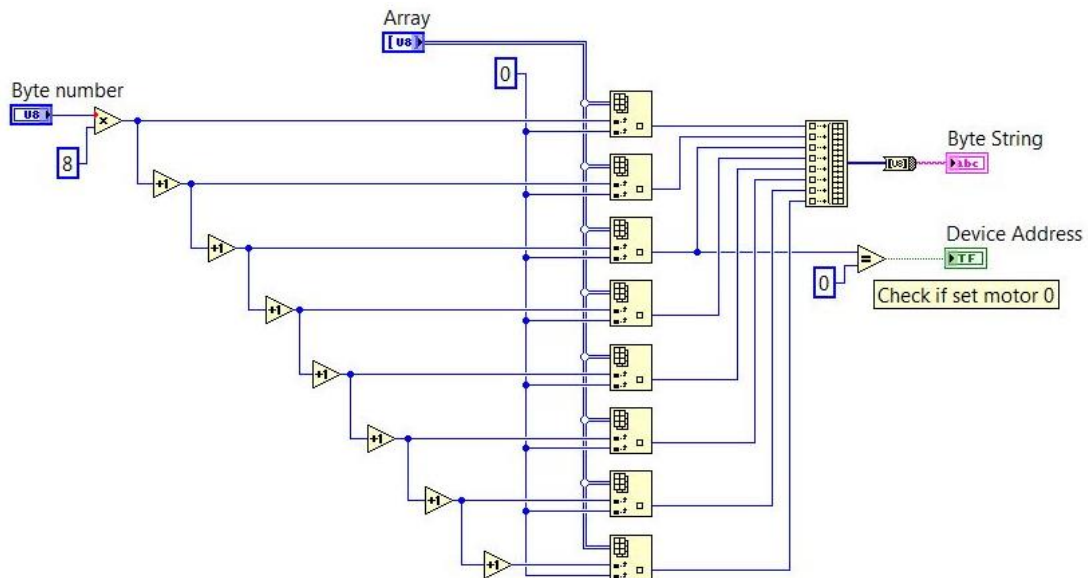



Figura 30 – *sR_ByteToString.vi* per il passaggio del dato da array di interi a stringa. Il check del motore avviene per comparazione tra ADR e l'indirizzo di uno dei motori (qui 0). Se il segnale *Device Address* risulta vero, allora il telegramma è stato inviato dall'azionamento 0, se falso dall'1.

All'uscita del case la stringa viene inserita nel registro per essere disponibile al ciclo successivo, e viene reso disponibile il segnale *Device Address*. Questi valori sono utilizzati dai due case successivi, posti in serie l'uno con l'altro. Essi sono del tutto identici, a meno dell'inversione dei casi vero/falso. Infatti i due *case* sono attivati dal segnale *Device Address* posto in prodotto logico con la verifica di ricezione. Questo permette di attivare due casi differenti per attuare due operazioni diverse cioè:

- mantenere il valore proveniente dal caso precedente se il dato non è riferito all'azionamento in questione;
- aggiornare il valore se il dato ricevuto deriva dall'azionamento considerato.

A livello software il case riesce a gestire la parte di selezione dei due casi qui presentati. L'aggiornamento dei dati è svolto tramite la procedura concatenazione del

Replaced Substring già descritta precedentemente. In questo caso però la prima stringa deriva dal caso di controllo sui byte letti. La seconda stringa, invece, deriva dal valore di un registro (diverso per ognuno dei due assi) che viene aggiornato ad ogni ciclo dal valore in uscita dal *case*. In questo modo si ottiene una stringa per ogni azionamento che contiene l'informazione secondo la codifica USS. Il fatto di sfruttare i registri permette di avere il valore aggiornato per un azionamento o di disporre dell'ultimo dato ricevuto, nel caso in cui il dato ricevuto ad un dato istante sia riferito ad un altro azionamento.

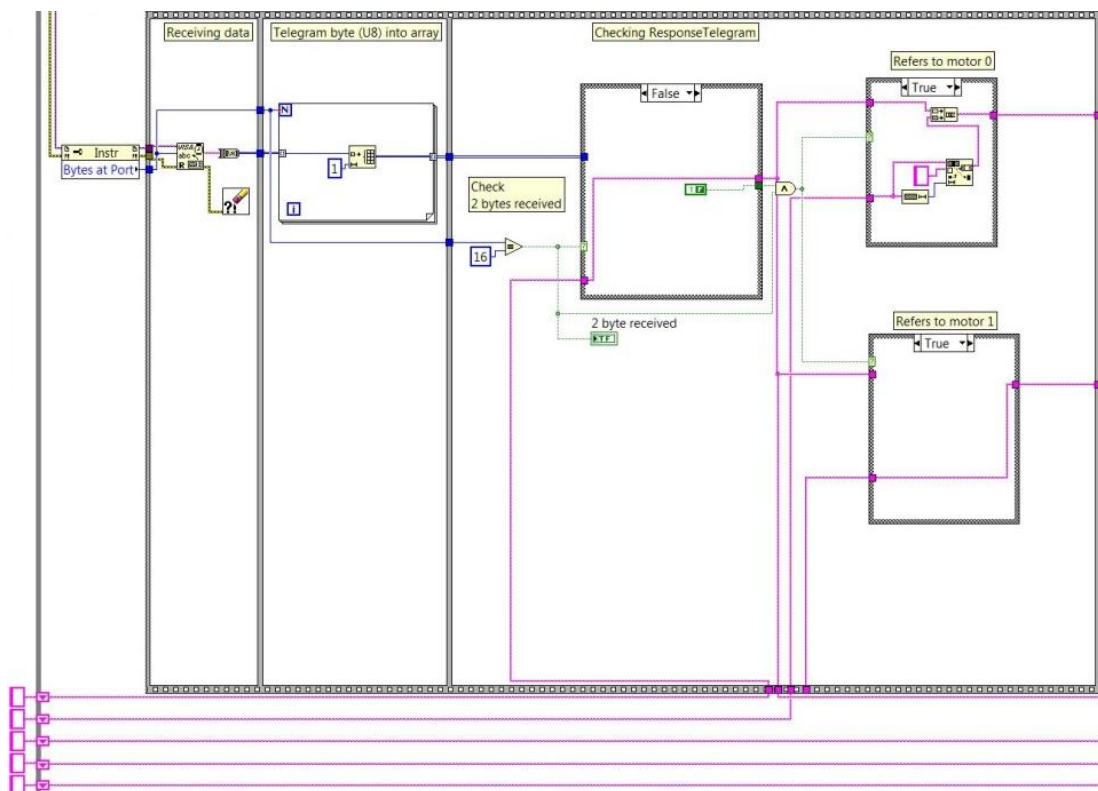



Figura 31 – Stralcio di codice rappresentante la prima parte di ricezione di *SiemensUSS.vi*. Il *Response telegram* viene letto dalla porta seriale e necessita di essere decodificato. Prima di affrontare la divisione dei pacchetti dati occorre capire da quale azionamento provenga tramite i case inseriti nel terzo frame della struttura sequenziale. I case identificati dalle etichette "motor 0" e "motor 1" si riferiscono agli azionamenti dei due assi, e si presentano identici a meno dell'inversione dei casi (vero e falso invertiti).

Dal frame successivo inizia la separazione dei vari pacchetti che compongono il *Response Telegram* duplicati identicamente per i due assi, a partire dalle rispettive stringhe derivanti dalla procedura fino a qui descritta. La stringa viene convertita in un vettore di interi grazie al comando *String To Byte Array* e da qui importata nella subVI *sR_ResponseTelegram.vi* che si occupa di dividere i vari PZD. I singoli byte

sono estrapolati dal vettore di interi U8 impostando al comando *Index Array* il valore d'indice del byte da estrapolare. I due byte di ogni PZD, assieme al carattere BCC, sono messi a disposizione all'uscita della funzione.

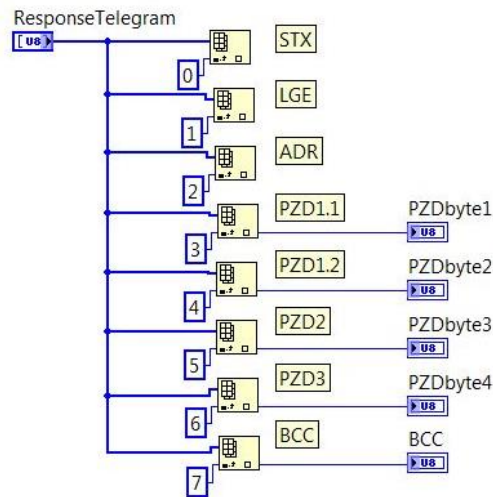



Figura 32 – *sR_ResponseTelegram.vi* per la divisione dei byte che compongono i caratteri di rete.

Il frame successivo è adibito al controllo sul carattere BCC, svolto dalla subVI *sR_BCCcheck.vi.* In ingresso alla funzione vengono collegati l'array contenente l'intero dato *Response Telegram* sotto forma di array, e il BCC ricevuto, opportunamente isolato dagli altri caratteri dalla funzione precedente. La VI fornisce in uscita una variabile, *BCC check*, che risulta vera se il BCC ricevuto e quello calcolato coincidono. Il controllo avviene comparando i due caratteri con il comando *Equal* posto in prodotto logico con il segnale derivante dal confronto (*NotEqual*) tra il conteggio di bit pari ad 1 e lo 0. Questo permette di sapere se l'array è composto da elementi tutti nulli, nel qual caso è sicuramente presente un errore di trasmissione. All'interno della subVI un *ciclo For* eseguito per l'intera lunghezza degli array, si occupa del calcolo di bit pari ad 1 andando a sommare ad un registro inizialmente nullo il valore di ogni bit. Per il calcolo del BCC si è fatto riferimento al protocollo USS che richiede il calcolo di uno XOR tra i singoli bit ed il valore del BCC calcolato dal bit precedente e qui implementato tramite un registro. Il valore uscente dal registro è il BCC calcolato e viene utilizzato per la comparazione.

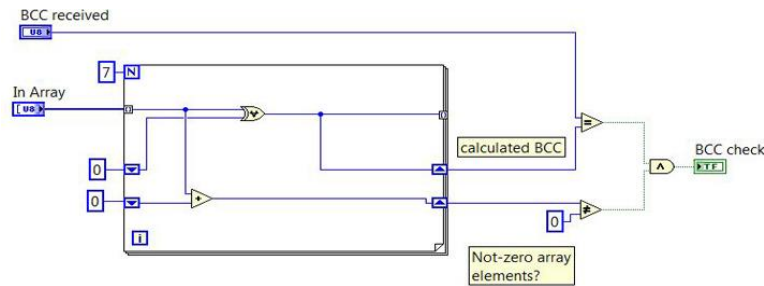



Figura 33 – *sR_BCCcheck.vi* per il calcolo e la verifica del carattere BCC.

Inoltre in questo frame si provvede ad esportare *Status Flag* nell’interfaccia cRIO tramite variabile globale (una per azionamento). Questa variabile, derivante dalla subVI di divisione dei byte, contiene informazioni sullo stato dei motori, ed in particolare:

0. Pronto all’accensione;
1. Pronto alla movimentazione;
2. Operazioni abilitate;
3. Anomalia attiva;
4. Bloccaggio inerziale attivo;
5. Bloccaggio rapido attivo;
6. Blocco inserzione attivo;
7. Avviso attivo.

Nei due flag successivi si utilizzano i due byte di PZD2 per il calcolo delle velocità degli assi derivante dall’azionamento. Specularmente a quanto è stato fatto nella fase di preparazione del *Main Setpoint* del *Task Telegram*, occorre unire i due byte del dato concatenandoli in un’unica stringa, solo dopo averli formattati ad interi ad 8 bit tramite il comando *Format Value*. La stringa così ottenuta viene portata all’ingresso di una struttura *case* attivata dal segnale di ricezione dei due byte. Nel caso falso, cioè quanto il dato non è stato interamente ricevuto, si porta all’uscita del case una seconda stringa derivante da un registro in cui è stata inserita l’ultima stringa utile. L’uscita del case, infatti, è collegata al registro di cui ne aggiorna il valore. Nel caso vero, invece, si utilizza la stringa derivante dal frame precedente e si esegue la procedura di concatenazione con il valore del registro, opportunamente elaborato,

come descritto per i casi dei frame precedenti. Il successivo ed ultimo frame utilizza le due stringhe derivanti dai rispettivi assi per ricavare il dato di velocità. Nella subVI *sR_PZDActualValue.vi* all'interno di un *ciclo For* avviene la conversione del dato effettuando un numero d'iterazioni pari al numero di caratteri di PZD2. Il case inserito realizza la sostituzione degli spazi vuoti (carattere 32 ASCII) con degli 0 (carattere 48 ASCII), lasciando il resto dei caratteri inalterati. Fuori dal case il numero decimale si ricava effettuando la conversione da ASCII, cioè sottraendo 48 ad ogni carattere. Infine si somma ad un registro, inizialmente nullo, il valore ricavato dal carattere moltiplicato per 2^x dove x è dato dalla posizione all'interno della stringa del carattere letto. La x più alta è per il bit più significativo. Il valore di x viene calcolato tramite la sottrazione tra il *numero caratteri-1* ed il *numero d'iterazione-1*, cioè *i*. Il valore in uscita dal ciclo rappresenta la velocità riportata in I16, che va convertita all'intervallo -6000RPM/+6000RPM dividendo il valore per 5.46133333333.

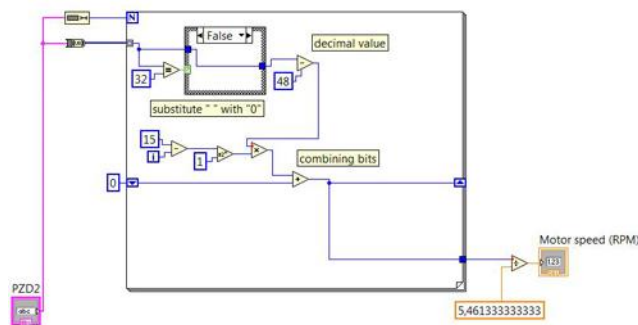



Figura 34 – *sR_PZDActualValue.vi* SubVI per il calcolo della velocità dell'asse partendo dalla stringa PZD2.

I valori di velocità sono quindi esportati nella VI principale operante su cRIO tramite variabili globali, *Motor Speed (RPM).*

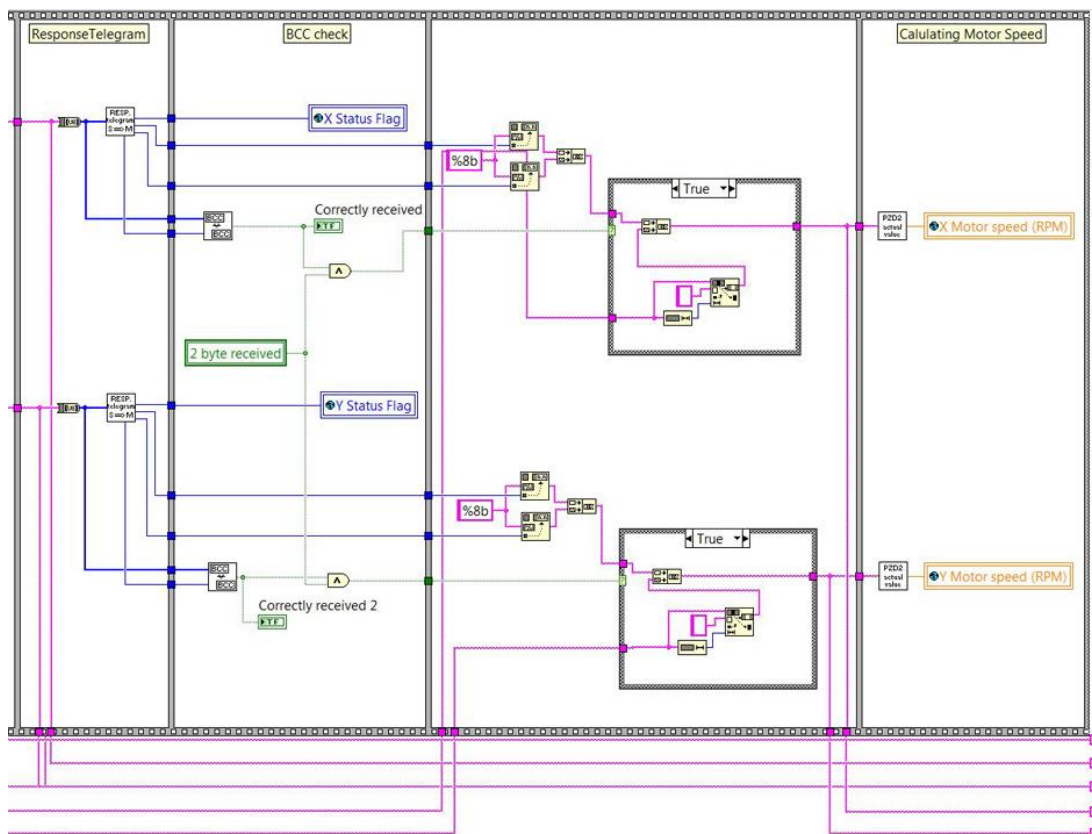



Figura 35 – Stralcio di codice rappresentante la seconda parte di ricezione di *SiemensUSS.vi.* Il *Response telegram* già diviso per i due azionamenti viene separato nei vari byte. Mentre i due byte di flag vengono esportati nell'interfaccia principale cRIO, i dati di velocità subiscono un ulteriore trattamento fino ad ottenere il valore di velocità, a sua volta esportato nella VI principale.

3.2.4. Sequenza di accensione

Per ottenere la corretta accensione dei motori occorre considerare i byte PZD1 del Task Telegram. Il protocollo prevede che l'accensione dei motori avvenga in passaggi separati. Poiché tutti i comandi sono attivabili soltanto se è già stato attivato il segnale di abilitazione operazioni, il primo segnale da abilitare sarà *Enable Operation*. Questo abilita tutte le operazioni consentite all'azionamento per l'utilizzo dei vari comandi. In seguito è necessario un controllo di anomalie ed eventualmente la loro tacitazione. In caso di anomalia attiva nessun comando di accensione può essere impostato poiché questo comporterebbe un malfunzionamento degli azionamenti. Il segnale di anomalia indica un qualsiasi errore – elettronico o fisico – dell'azionamento. Generalmente l'anomalia è generata dal timeout dell'azionamento che comporta la disconnessione dello stesso, nel peggiore dei casi può essere causato da un bloccaggio fisico del rotore. Per questo la tacitazione delle anomalie è un

passaggio critico in quanto è possibile che il software non sia in grado di resettarle, ed in tal caso è richiesto l'intervento di manutenzione dell'operatore. Qualora non vi siano anomalie attive, l'accensione è completata con il comando *Accensione* dei motori.

La sequenza di accensione deve dunque tenere conto di:

- sequenza di attivazione dei segnali;
- tempo di lettura da parte dell'azionamento;
- verifica di acquisizione dei segnali da parte degli azionamenti.

La sequenza di attivazione già descritta richiede il controllo da parte di ogni variabile critica di una struttura *case*, come specificato in **Figura 36**.

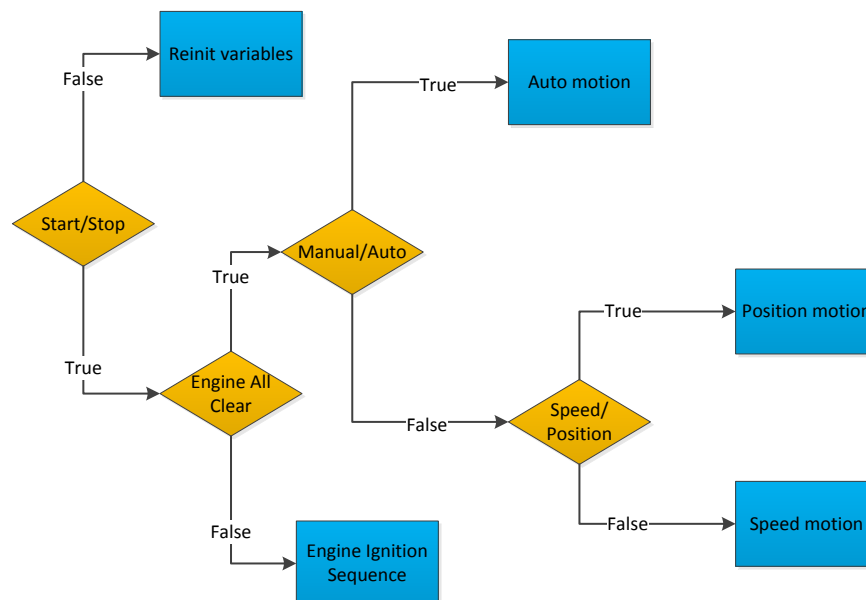



Figura 36 – Diagramma logico della sequenza di attivazione ed utilizzo dei comandi motore.

Per quanto riguarda il tempo di lettura da parte dell'azionamento si ritiene che un intervallo di tempo tra l'attivazione di un segnale ed il successivo debba essere almeno due volte il tempo di trasmissione USS, per essere certi che il dato giunga almeno una volta. Per questo il tempo di attesa è impostato a 70 ms.

Per la verifica dell'acquisizione del segnale da parte degli azionamenti si ricorre alla creazione di un'apposita variabile *ReadyToSwitchOn*. Questo segnale indica che l'azionamento non presenta anomalie, blocchi attivi ed è dunque pronto

all’accensione. Il segnale è generato utilizzando gli *Status Flag* di entrambi gli assi. Su ogni asse una somma logica tra le variabili *Blocco inserzione* e *Pronto all’accensione* (rispettivamente bit 0 e 6) generale il segnale *FatalError* che può essere considerato come un indicatore di errore. Un ulteriore passaggio di unione dei segnali dei due assi tramite un prodotto logico permette di ottenere la variabile *ReadyToSwitchOn*. In tal modo la variabile è vera soltanto se su entrambi gli assi non sono presenti errori. Un secondo segnale chiamato *EngineAllClear* è ottenuto per prodotto logico tra i segnali di motori accesi *On/Off* ed *OperazioniAbilitate* (bit 1 e 2) di ogni asse ed il segnale *ReadyToSwitchOn* (**Figura 37**). Questo segnale è vero soltanto nel caso in cui i motori sono accesi e non vi sono anomalie.

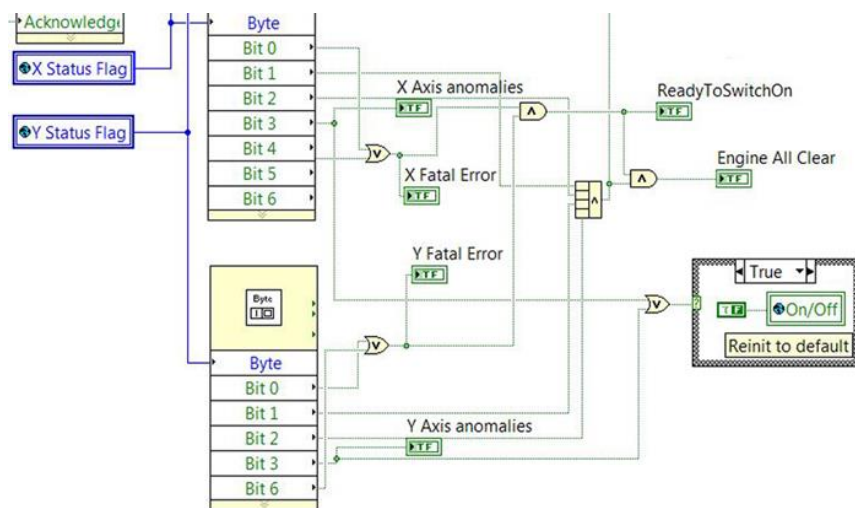



Figura 37 – Generazione del segnale *ReadyToSwitchOn*.

La sequenza di accensione si trova all’interno del *case False* di *EngineAllClear*, mentre nel *case True* è presente tutto l’algoritmo di controllo del sistema. Un *case* più esterno è attivato dalla variabile *Start/Stop* che rappresenta la richiesta dell’utente di accensione dei motori. Nessuna operazione è consentita all’utente in caso di motori spenti. Nel caso di condizione *Start/Stop* falsa, cioè nella fase di spegnimento, si inizializzano tutte le variabili di velocità, impostando a 0 il setpoint di velocità e le variabili di posizione ponendole alla posizione attuale letta da encoder per evitare problemi in fase di riaccensione. Un ulteriore variabile, *stopped*, è impostata a vero per potere in seguito reinizializzare il blocco *PID Rate Limiter* e le variabili di velocità rilevata del motore nella prima iterazione di riaccensione, cioè quando il dato non è ancora aggiornato dalla trasmissione USS. Questi accorgimenti sono

necessari per evitare di incorrere in accelerazioni troppo repentine e/o di movimenti inaspettati in fase di riaccensione del motore. Nel caso in cui la variabile *Start/*Stop sia vera si passa al controllo sulla variabile *Engine All Clear* che identifica lo stato di operatività del sistema motori.

La sequenza di accensione in sé è costituita da una struttura sequenziale. Il primo frame si occupa dell'abilitazione delle operazioni impostando a vero la variabile globale *EnableOperation*. Segue un frame di attesa di un tempo di 70 ms (più di due volte il tempo del ciclo di comunicazione USS) per attendere la ricezione del comando. Nell'ultimo frame si effettua un controllo di anomalie in un *ciclo While* e la loro tacitazione, se necessaria, avviene automaticamente collegando la variabile globale *Ack Anomalies* all'uscita della variabile *Axis Anomalies*, derivante da un ciclo parallelo. Così facendo ad ogni iterazione si effettua un controllo sulle anomalie, ed in caso vi siano si provvede alla tacitazione. L'uscita da tale ciclo avviene per assenza di anomalie, ricavata con un NAND tra le anomalie dei due assi, o per un basso numero di iterazioni che generano un segnale di uscita tramite un comparatore posto sull'indice *i*. I due segnali di uscita, posti in somma logica, permettono di uscire dal ciclo evitando di incorrere in un loop infinito in caso di impossibilità di tacitazione. All'uscita del ciclo il segnale derivante dal NAND sopra descritto, che è vero se non ci sono anomalie, permette l'accesso ad una struttura *case* dove si procede all'accensione dei motori. Nel caso vero (assenza di anomalie) si fornisce l'abilitazione del segnale *On/Off* nel primo frame di una struttura a due passaggi. Dopo l'accensione si attende un periodo di sicurezza per essere certi che il motore riceva il comando.

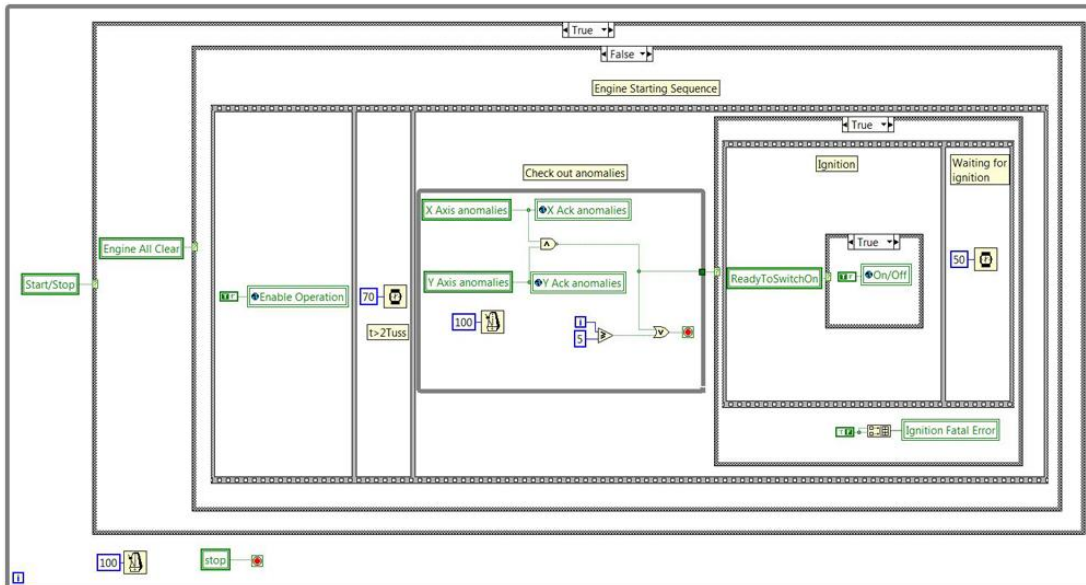



Figura 38 – Sequenza accensione motori. Nella struttura sequenziale del ciclo interno sono ben visibili tutte le fasi descritte per la tacitazione delle anomalie e l'invio del comando di accensione.

3.2.5. Preparazione pacchetti dati cRIO

L'interfaccia cRIO deve inviare all'interfaccia utente tutti quei dati di funzionamento del sistema. Questi sono divisi in due pacchetti chiamati *Operation Data* (ID=0) e *System Data* (ID=3)[12]. Il primo, *Operation Data*, è costituito da due pacchetti distinti, uno di flag ed uno di misure e setpoint. In un *ciclo While* è presente una struttura sequenziale a tre frame. Nel primo i dati vengono raccolti e subito passano al secondo, in modo che tutti i dati raccolti siano riferiti ad un istante comune. Nel secondo frame avviene l'assemblamento inserendo all'interno di array i vari dati che compongono il pacchetto. Si passa così al terzo frame nel quale la richiesta di trasmissione è inserita in coda. Per l'inserimento della richiesta in coda è necessario prima verificare che non sia già presente un'altra richiesta per lo stesso ID eseguendo un *ciclo While* che analizza ogni richiesta confrontando ogni elemento della *RT queue* con il *cluster* di ID e subID di riferimento. In uscita dal ciclo si accede ad una struttura case che inserisce in coda la richiesta con la funzione *Enqueue Element at Opposite End* nel caso in cui una richiesta per lo stesso pacchetto non sia già presente in coda. In caso opposto il *case* risulta vuoto e non esegue istruzioni. L'intero ciclo di impacchettamento è stato temporizzato secondo la frequenza di aggiornamento di 20

[12] Riferimento ATTP: si veda capitolo quarto.

Hz, a 100 ms. Una frequenza maggiore risulterebbe uno spreco di risorse poiché aggiornerebbe dati uguali, dal momento che anche il ciclo di acquisizione da FPGA è temporizzato a 100 ms.

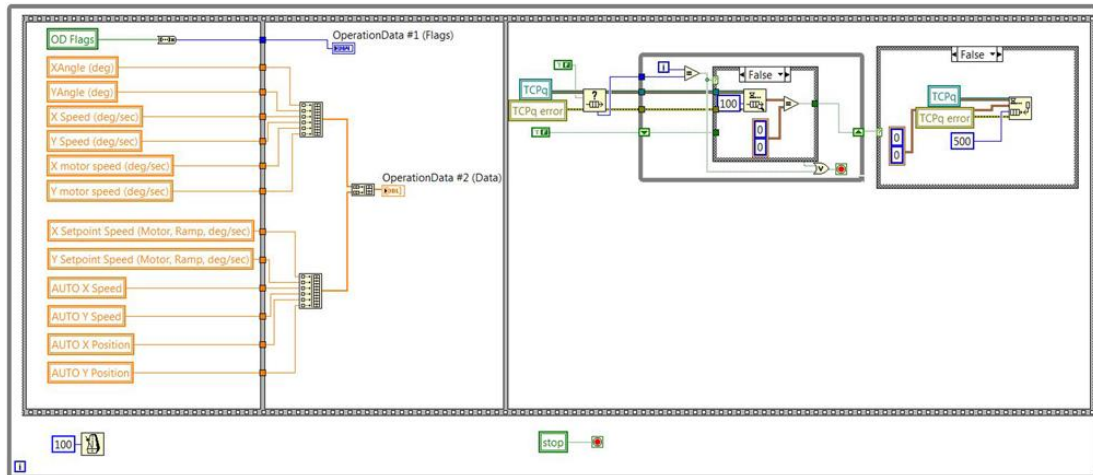



Figura 39 – Impacchettamento dei dati *Operation Data*. Da notare che i dati presenti in *OperationData #2 (Data)* sono costituiti da due array, uno di dati misurati da encoder e l'altro di setpoint imposto, che vengono poi uniti. Perché il comando *Build Array* svolga questa funzione è necessario spuntare l'opzione *concatenate inputs* dal menù a tendina onde evitare che il blocco crei un array bidimensionale anziché unire i due vettori.

Il secondo pacchetto è quello dei dati di sistema che è composto da tre pacchetti: il tempo del sistema cRIO, i dati meteo e i dati GPS. Tutti questi dati hanno una frequenza di aggiornamento molto bassa poiché sono variabili di interesse secondario e comunque dalle variazioni molto lente nell'unità di tempo. Il timing scelto per il ciclo di trasmissione dei *System Data* è dunque di 1 minuto, ovvero 60000 ms. All'interno del ciclo una struttura sequenziale raccoglie il dato dell'ora di sistema e nel secondo frame inserisce la richiesta di trasmissione in coda. In questa versione di ALMATracker non sono disponibili i dati meteo, dunque non vengono mai aggiornati. Inoltre dato il tempo di aggiornamento piuttosto dilazionato, in questo caso non si esegue il controllo della presenza di altre richieste per la stessa coppia ID-subID, supponendo che in tale tempo il dato sia stato certamente inviato.

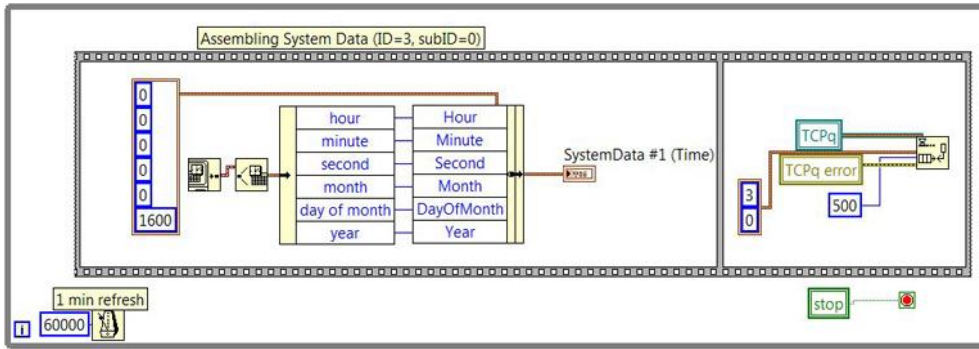



Figura 40 – Ciclo di preparazione dei *System Data*.

3.3. INTERFACCIA CLIENT

L'interfaccia client è quella parte del software eseguita su PC Windows per la comunicazione tra cRIO e utente. L'utente deve potere:

- Visualizzare parametri di funzionamento del sistema;
- Visualizzare dati provenienti dal sistema da dispositivi ausiliari (es: GPS, meteo, ecc…);
- Controllare il sistema.

Tutti i comandi destinati alla cRIO sono impostati direttamente dall'utente. In background una parte dell'applicazione procede all'aggiornamento continuo dei dati provenienti dalla cRIO e mostra a schermo i valori. Lo schema logico riportato in **Figura 41** riassume i comportamenti dell'interfaccia client.

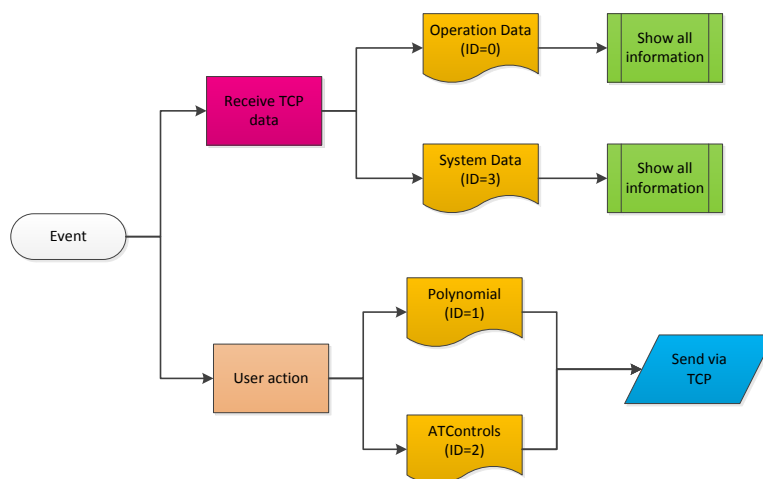



Figura 41 – Schema logico della VI del Client in cui si evidenziano le funzioni da svolgere a seconda degli eventi.

L’interfaccia necessita quindi di due cicli paralleli con due differenti compiti da svolgere, uno di ricezione del dato e uno di trasmissione. Per quanto riguarda il ciclo di visualizzazione nel dettaglio si rimanda al capitolo quarto. I dati ricevuti sono visualizzati da indicatori che permettano una corretta e istintiva lettura sull’interfaccia da parte dell’ utente. I dati in trasmissione sono invece inseriti in coda dal ciclo di controllo tramite variabili locali. Il ciclo di controllo ha come scopo quello di aggiungere in coda una richiesta di aggiornamento ogni qualvolta un valore di controllo venga modificato dall’utente. Tale comportamento si ottiene facilmente con l’utilizzo di una coda e di variabili locali che vengono aggiornate con una frequenza di 20 Hz per essere pronte, in caso di modifica, ad andare ad aggiornare il valore su cRIO.

3.3.1. Inizializzazione dei comandi

L’inizializzazione dei dati si ottiene inserendo tutto il codice all’interno di una struttura sequenziale ed aggiungendo un primo frame riservato al setting dei dati di default tramite degli *Invoke nodes*.

Numerosi sono inoltre gli eventi che sono stati considerati in fase di test e che hanno portato allo sviluppo di una serie di accorgimenti particolari per migliorare il funzionamento del sistema finale. Il primo riguarda il caso di spegnimento dei motori o di disconnessione della rete. In questo caso occorre reinizializzare le manopole che forniscono i setpoint di posizione e velocità per evitare che riaccendendo i motori o riconnettendo la rete si abbia un’accelerazione inaspettata del sistema. Si utilizza un *case* attivato da almeno uno tra i segnali *Start* o *TCP connection* falsi. All’interno della struttura si provvede a reinizializzare le variabili di velocità – poste a 0 – e le variabili di posizione – poste al valore attuale letto da encoder. Inoltre si pone *Parking* a default, poiché il comando dovrebbe essere trattenuto con l’azionamento acceso se fosse necessario mantenere la posizione di minima resistenza. Anche *Start/Stop* viene reimpostato a default, coerentemente con quanto viene fatto da lato RT per poter spegnere l’azionamento. Il precedente ragionamento non viene effettuato in caso di tracking automatico. Se il sistema sta eseguendo un tracking di un oggetto orbitale anche in caso di disconnessione

continua il proprio moto, questo per evitare interruzione nella ricezione del segnale da parte della stazione di terra.

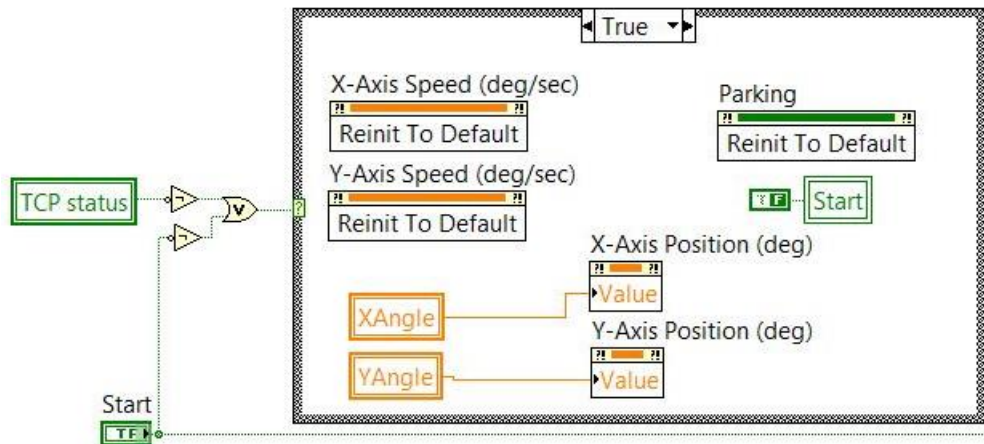



Figura 42 – Reinizializzazioni di angoli e velocità in seguito a caduta della connessione o spegnimento motori.

Il secondo accorgimento riguarda la modalità di *Parking*, per cui viene utilizzato una semplice movimentazione in posizione. Se il comando viene inserito si attiva un *case* in cui si imposta il comando di movimentazione manuale in posizione, imponendo il setpoint prefissato nel *Tab Parking*.

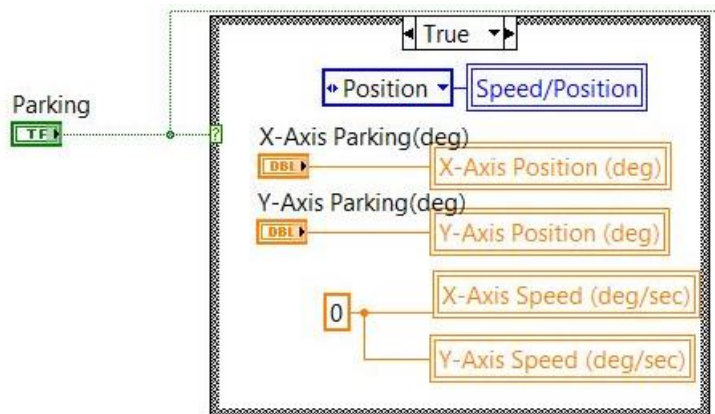



Figura 43 – Modalità di parking. Anziché richiedere un vero e proprio parking è l'interfaccia utente che provvede ad impostare la posizione di sicurezza tramite la movimentazione in posizione.

3.3.2. Acquisizione delle polinomiali

Poiché la generazione della polinomiale attualmente è a carico di un programma esterno, Matlab, è necessario importare i valori dei coefficienti delle polinomiali da file esterni. La soluzione adottata è composta da una generazione di due file di testo da parte di Matlab con i due vettori “alphafit” e “betafit” contenenti rispettivamente i

coefficienti della prima polinomiale (asse X) e quella della seconda (asse Y) in ordine di grado decrescente. Il codice utilizzato permette la scrittura su file delle due matrici in colonna secondo ordine di grado crescente, così come richiesto dal software ALMATracker. La funzione Matlab fliplr permette appunto di rovesciare l’intero vettore e crearne uno che risponda a tali requisiti. Di seguito si riporta il codice che genera i due file FILEalpha.txt e FILEbeta.txt.

```
format long;

f1 = fopen('FILEalpha.txt','w');

fprintf(f1,'%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f',
fliplr(alfafit));

fclose(f1);

f1 = fopen('FILEbeta.txt','w');

fprintf(f1,'%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f',
fliplr(betafit));

fclose(f1);
```

Una volta che i file sono stati creati, è possibile importarli[13] sul software ALMATracker tramite il tasto *Change polynomial*. Questo apre una finestra di popup dove è possibile eseguire il puntamento dei file tramite *file dialog*. Con il pulsante di conferma si attiva la funzione *Read From Spreadsheed File* per modificare i valori dei coefficienti di due array di double e inserire in coda la richiesta di aggiornamento via TCP.

---

[13] Prima di importare i file in ALMATracker è importante tenere conto delle differenti notazioni dei due software. Infatti per la separazione dei decimali Matlab utilizza il punto mentre LabView la virgola. E’ quindi necessario sostituire tutti i punti con delle virgole (ad esempio con blocco note *Modifica>Sostituisci*> e si seleziona “.” con “,”).

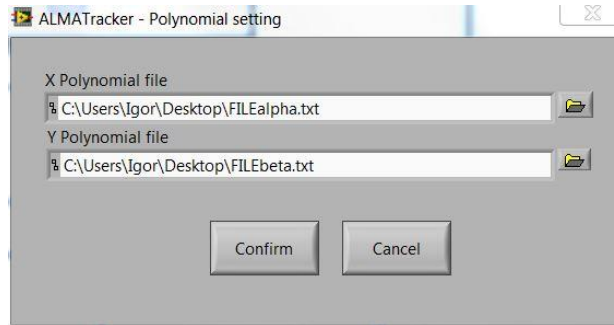



Figura 44 – Interfaccia di aggiornamento dei coefficienti della polinomiale

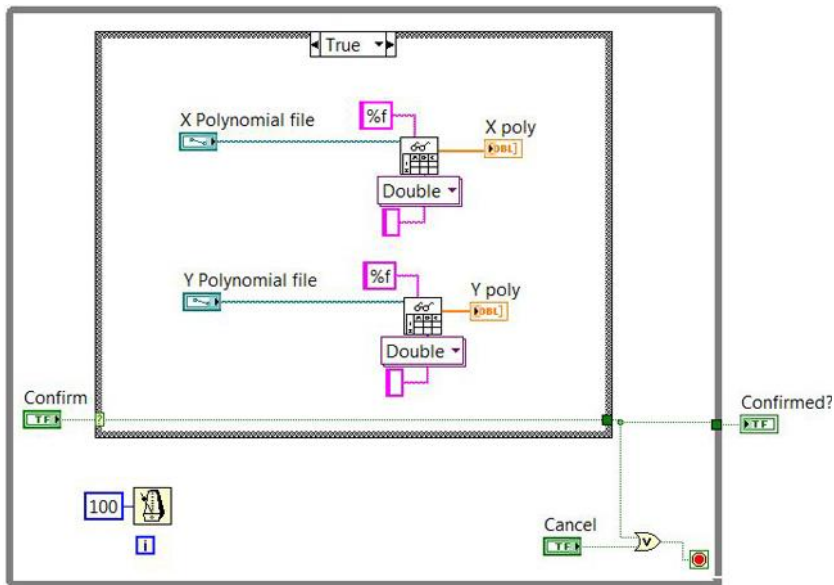



Figura 45 – *Block diagram* dell'interfaccia di aggiornamento dei coefficienti della polinomiale

3.3.3. Preparazione pacchetti dati client

I dati che l'interfaccia client si occupa di inviare alla cRIO riguardano le polinomiali (ID=1) ed i controlli ALMATracker (ID=2)[14]. Questi dati vengono impacchettati, associati ai propri identificativi di trasmissione e inseriti all'interno della coda *RT queue* in maniera del tutto analoga a quanto avviene nell'interfaccia cRIO. Entrambi i pacchetti non sono aggiornati a tempo ma ogni volta il dato subisce una modifica. Per questo tutto l'aggiornamento dati avviene in un unico *ciclo While* temporizzato a 50 ms. Un timing inferiore risulta di scarsa efficacia poiché il tempo di aggiornamento dei comandi effettuato manualmente difficilmente scende sotto tale soglia. Inoltre occorre evidenziare che, per lo stesso motivo, non è stato ritenuto

[14] Riferimento ATTP: si veda capitolo quarto.

necessario adottare provvedimenti per la verifica della presenza simultanea di due richieste identiche in coda.

Il pacchetto dati delle polinomiali è aggiornato soltanto quando il segnale *ChangePolinomial* è vero. Infatti questo segnale deriva dalla finestra popup in cui l'utente ne richiede la modifica ed attiva un *case* dove la polinomiale viene aggiornata. In questo modo il dato è aggiornato automaticamente ed è necessario soltanto inserire in coda la richiesta di trasmissione. Nel ciclo di aggiornamento si compila tale richiesta, tramite un case collegato alla variabile *ChangePolinomial* (se risulta vera, la richiesta viene compilata). E' inoltre necessario reinizializzare la variabile di controllo al valore falso dopo aver compilato la richiesta.

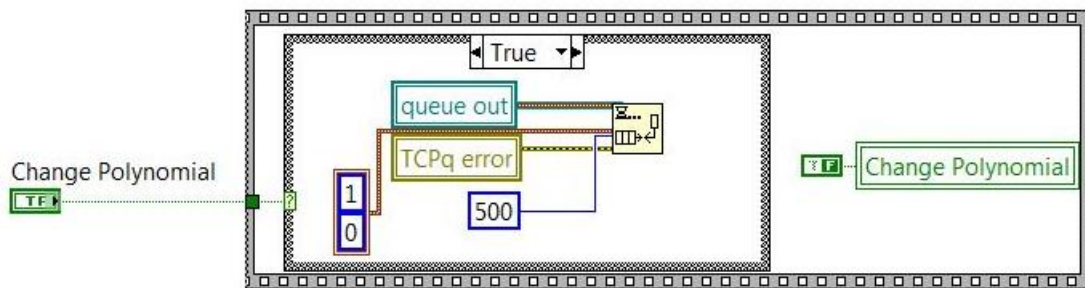



Figura 46 – Aggiornamento della richiesta di trasmissione delle polinomiali.

Diversamente avviene per i pacchetti dati facenti parte di *ALMATracker Controls*. Per ogni pacchetto dell'ID 2 è necessario, dopo la raccolta dei dati, confrontare il pacchetto attuale con quello precedente tramite la funzione comparativa *NotEqual* (≠). La successiva struttura *case* deve provvedere a compilare la richiesta con ID e subID riferiti a quel specifico DP nel caso vero, cioè quando il dato è cambiato.

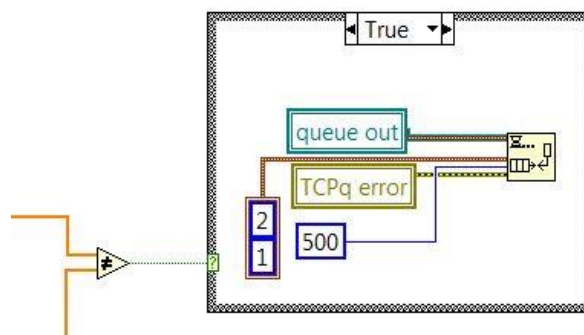



Figura 47 – Esempio di richiesta di aggiornamento del DP *Setpoint* di *ALMATracker Control*. I due cavi provenienti da sinistra derivano l'uno dal dato attuale, l'altro dal dato precedente presente nel registro. Poiché si

tratta di array è necessario portare il *comparison mode* dell'operatore *NotEqual* in modalità *compare aggregates* dal menù a tendina.

I dati da assemblare vengono raccolti in maniera speculare per i DP 0 ed 1, entrambi creati con la funzione *Build Array* a partire dai dati necessari. Per il pacchetto dell'ora di sistema invece la richiesta è attivata, come avviene per la polinomiale, dal tasto *ConfirmSync*, e dunque ne condivide la medesima procedura.

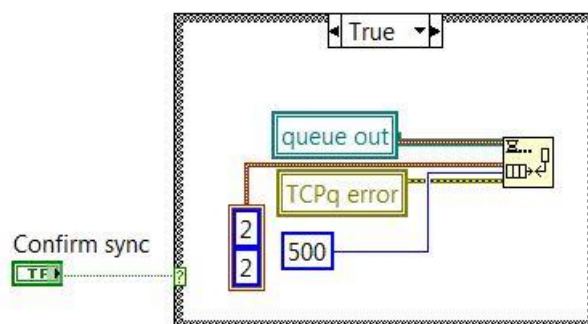



Figura 48 – Aggiornamento della richiesta di trasmissione di sincronizzazione ora.

Per l'aggiornamento dell'ora è invece necessario prima di tutto leggere il dato di tempo in secondi dall'orologio del PC. Questo viene poi convertito in *cluster* tramite la funzione integrata *Second to DateTime* impostando l'ora come UTC. Il dato viene poi filtrato tramite un passaggio svolto dalle funzioni *Unboundle by Name* e *Boundle by Name*, utile per eliminare i dati al di sotto del secondo. Il dato completo è infine spedito inserendo la richiesta in coda.

3.3.4. Interfaccia grafica

Per la compattazione a livello grafico del *Front Panel* sono stati utilizzati alcuni *Tab Container* in modo tale da aver a disposizione tutte le informazioni contenute in una schermata selezionabile.

Nel campo di movimentazione automatica sono stati inoltre inseriti degli indicatori che mostrano le in tempo reale i setpoint di velocità e posizione comandati dal modulo RT in riferimento alle polinomiali inserite.

Per quanto riguarda i controlli del sistema, qui sono inseriti gli indicatori di posizione GPS, condizioni atmosferiche che non sono stati previsti in questa sede, ma già predisposti sia in visualizzazione che in trasmissione dati TCP. In questa sezione

sono presenti anche i comandi di aggiornamento del time della cRIO e il setting dei range di collisione.

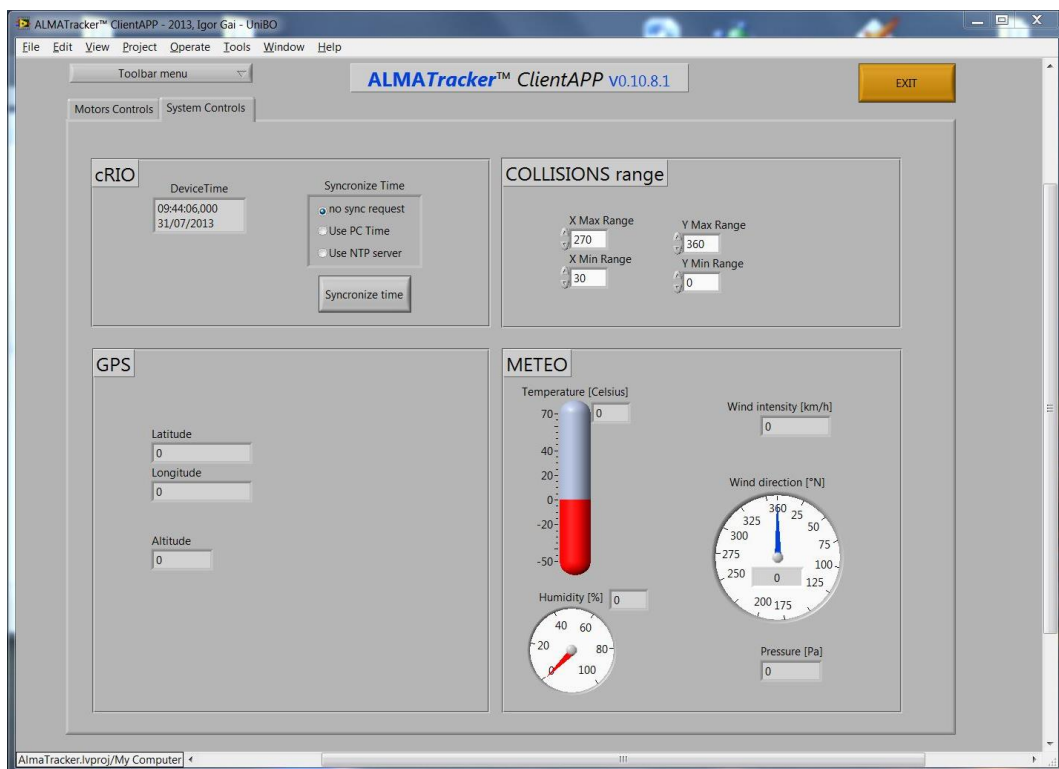



Figura 49 – Screenshot dell'interfaccia grafica.

Capitolo Quarto

COMUNICAZIONE DATI VIA TCP/IP

La comunicazione dei dati tra cRIO e PC client richiede una buona rapidità di aggiornamento di diversi elementi e la possibilita di link a distanza ed è per questo che si è resa necessaria una connessione via ethernet. Tra i vari metodi di comunicazione è stato scelto il protocollo TCP/IP (Trasmission Control Protocol) sia per la semplicità d'utilizzo che per la possibilità di accedere al sistema rapidamente e ovunque, disponendo di una connessione internet[15]. La connessione TCP richiede l'utilizzo di una macchina server ed una o più client. Data l'architettura del progetto e l'esigenza di poter comunicare con il dispositivo cRIO con PC differenti è stato adottato un sistema in cui la cRIO è il server della connessione TCP (con IP fisso) e il client è un qualsiasi PC operante con sistema Windows. La differenza tra server e client sta nel fatto che il client può disconnettersi dal sistema senza generare anomalie e/o interruzioni di task (supponendo di essere in modalità automatica) mentre il server deve sempre essere operativo e mantenere aperto il canale di comunicazioni per eventuali client. Considerata inoltre la possibilità di avere client anche differenti[16], lo scambio dati deve avvenire secondo uno schema fisso deciso in precedenza. Per questo si è sviluppato un protocollo proprio (ATTP) e lo sviluppo dell'algoritmo stesso è stato eseguito in modo tale da essere bidirezionale e poter usare lo stesso codice sia da lato client che server. La differenza tra nei blocchi di trasmissione non sta nel codice di comunicazione ma nei dati scambiati con le VI di lettura e trasmissione, poiché il dato da trasmettere o ricevere è selezionato in modo differente nel server e nel client.

---

[15] E' necessario che la cRIO sia collegata ad una rete internet, che il router abbia aperto la corrispondente porta di comunicazione, e che l'utilizzatore conosca l'indirizzo IP pubblico del dispositivo.

[16] L'accesso al sistema deve poter avvenire da più di un PC, ma non in maniera simultanea.

4.1. PROTOCOLLO ATTP (ALMATRACKER TCP/IP PROTOCOL)

AlmaTracker Tcp/ip Protocol denota il protocollo di comunicazione TCP/IP sviluppato per questo sistema. ATTP è stato pensato modulare ed espandibile con la possibilità di aggiungere in ogni momento pacchetti dati di scambio. I pacchetti di dati sono classificati da un ID che identifica il gruppo primario (es: dato di funzionamento, controllo, dato di sistema…) e il verso del flusso del dato (da cRIO a client o viceversa) e da un subID che identifica, all'interno di uno stesso gruppo, la categoria del dato. Ogni pacchetto (DataPackage o "DP") con stesso ID e subID è caratterizzato da dati che viaggiano nella stessa direzione e che hanno stessa struttura (array di boolean, cluster, double, ecc…). I dati da trasferire sono setpoint dell'utente, variabili impostate dalla cRIO, letture di posizione, letture divelocità e indicatori di stato dei motori. La descrizione completa di tutto il protocollo è stata inserita in appendice, in ottica di fungere da manuale tecnico in un futuro utilizzo del sistema ALMATracker da parte di terzi. Alcuni pacchetti sono volutamente lasciati liberi per lasciare la possibilità di essere espansi sia in fase di sviluppo e affinamento, svolta in questa sede, sia per eventuali sviluppi futuri.

4.2. CICLO DI RICEZIONE

I dati ricevuti via TCP sono divisi in più trasmissioni. Le prime due trasmissioni sono di lunghezza definita ed identificano due numeri interi (in formato intero I32) che rappresentano rispettivamente ID e subID del dato che si sta per ricevere. Tramite gli identificativi ricevuti si accede ad una ricezione dei dati successivi che avviene in maniera distinta grazie all'utilizzo di due strutture *case* annidate in grado di distinguere a priori il dato ricevuto implementando differentemente la ricezione secondo il protocollo derivante dai vari identificativi. In tal modo si possono aggiornare i dati non appena ricevuti, andando a modificare la variabile corrispondente al dato identificato dall'ATTP. La ricezione dei dati avviene tramite il comando *TCP Read*, al quale occorre fornire l'ID di connessione, e il numero di byte da leggere e restituisce il dato letto sotto forma di stringa ASCII. Una caratteristica comune per la ricezione è che per ogni identificativo sono presenti uno o più *DataPackage* (DP), letti separatamente secondo una procedura fissa: viene letta dapprima la lunghezza del *DataPack* (LOP, di lunghezza prestabilita in ATTP) e

successivamente il DP di lunghezza nota da LOP. Il dato ottenuto viene poi riportato al tipo originale eseguendo il casting al tipo di dati specificato dal protocollo, grazie alla funzione *Typecast*.

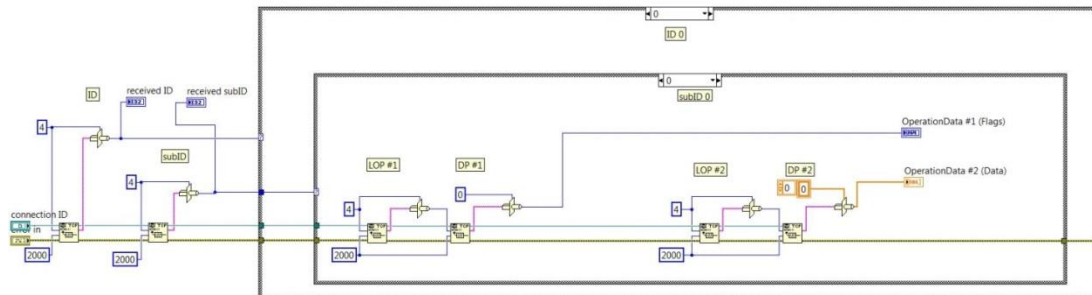



Figura 50 – VI di ricezione dati da TCP (*s_TCPread.vi*). Da notare che le letture di ogni blocco di dati avvengono separatamente per sapere in anticipo, prima del termine del pacchetto dati completo, cosa si va a leggere. Questo permette di aggiornate le variabili durante la fase stessa di ricezione, evitando una post elaborazione che sarebbe alquanto più lunga e complessa. E’ di estrema importanza prestare la massima attenzione alla rappresentazione delle costanti su cui si esegue il *Typecast*; e.g.: se in fase di trasmissione la conversione è avvenuta usando un U8, non è possibile in ricezione utilizzare, ad esempio, una U16. Infatti in quanto si richiederebbero più bit di quanti effettivamente inviati e si genererebbe un errore silente causato da uno slittamento dei dati tra i vari pacchetti.

In fase di ricezione occorre inoltre tenere conto che il ciclo aggiorna un solo DP per volta restituendo tutti gli altri DP nulli. Per questo si è pensato di aggiornare un dato soltanto se ID e subID indicano che quel dato è stato letto. Questa routine si ottiene inserendo un *Select* che riceve come riferimento la comparazione tra ID e subID in ingresso e quelli dello specifico DP, restituendo il nuovo valore letto se la comparazione ha esito positivo oppure il valore letto da *shift register* in caso di esito negativo. Nella versione finale, tuttavia, si è giunti ad un algoritmo di aggiornamento più compatto e con un minor consumo di risorse. Specularmente a quanto si osserva nella subVI *TCPread*, i valori vengono aggiornati solo all’interno del corrispondente caso di ID e subID di due strutture *case* annidate, comandate appunto dagli identificativi di trasmissione. In tal modo vengono aggiornati i valori ricevuti, mantenendo inalterati tutti gli altri, con il minimo costo in termini di calcolo e tempo. La ricezione dei dati restituisce quindi dei valori derivanti da un’altra macchina filtrati in maniera differente a seconda che si consideri la cRIO o il PC. In particolare si andranno ad aggiornare soltanto i valori che, da protocollo, si presume arrivino dall’altra macchina, trascurando completamente di aggiornare dati il cui ID è

incompatibile con la ricezione. Ad esempio un PC non deve aggiornare dati il cui ID indica in arrivo da un PC perché questi sarebbero sicuramente derivanti da un errore di comunicazione dati. Sia nella subVI che nel case di aggiornamento è previsto un *void case* identificato dall'ID 50, che viene utilizzato qualora l'aggiornamento dei dati non sia stato richiesto, e quindi nessuno dei precedenti valori debba essere sovrascritto.

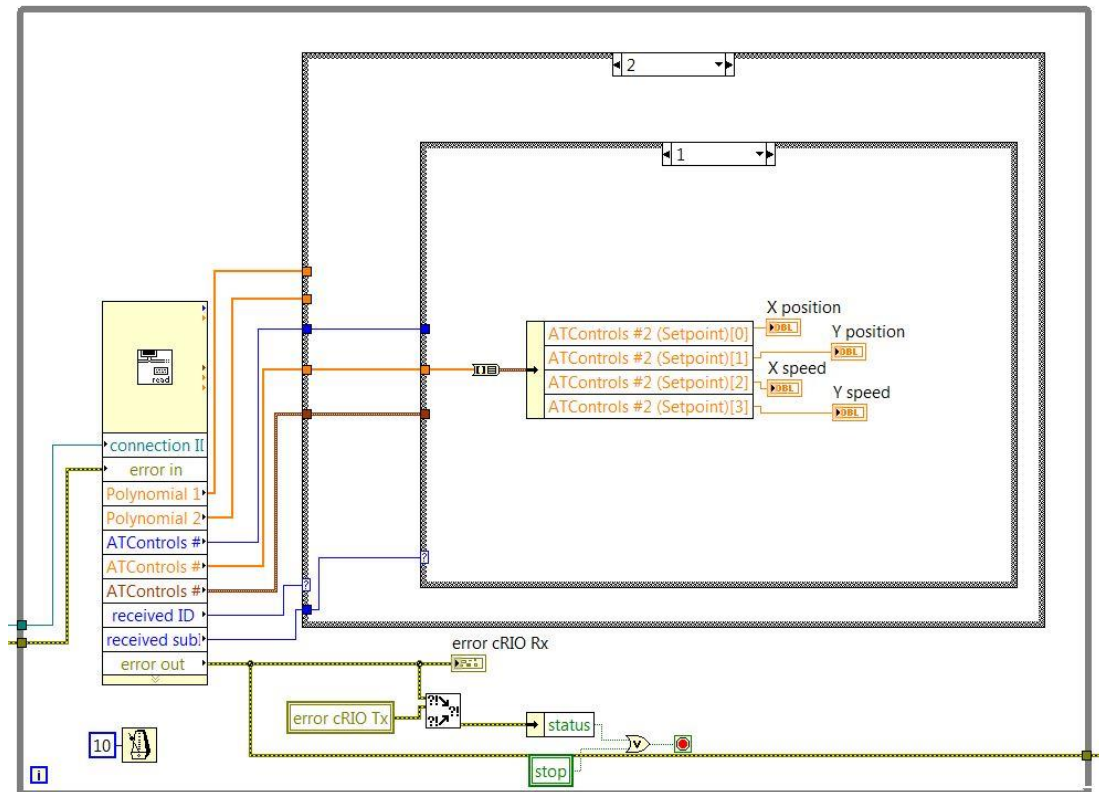



Figura 51 – Ciclo di aggiornamento dei dati provenienti dalla connessione TCP. In questo caso è esaminata la parte cRIO che va quindi ad aggiornare i DP di ID 1 (polinomiali) e 2 (comandi). In basso invece è presente la gestione degli errori e del segnale di arresto manuale che devono essere in grado di fermare l'intero ciclo in caso di disconnessione, rendendo possibile la ricerca di una nuova connessione disponibile.

4.3. CICLO DI TRASMISSIONE

I dati sono spediti in maniera del tutto speculare alla procedura di ricezione. ID e subID vengono inviati nelle prime due trasmissioni, successivamente vengono spediti tutti i dati richiesti (a seconda degli identificativi) ognuno preceduto dalla propria lunghezza. Ogni tipologia di dato (boolean, double, ecc…) è diviso in DP differenti per comodità di trasmissione dati. Infatti per la trasmissione in un

pacchetto TCP i dati non possono essere trasmessi come valori numerici ma come stringhe di caratteri, e se i dati sono omogenei (cioè dello stesso tipo) è possibile la conversione da stringa a double utilizzato la funzione integrata *Typecast* che esegue la conversione in stringa. Il dato ottenuto viene trasmesso e in fase di ricezione può essere riottenuto eseguendo un cast al tipo di variabile del DP (da protocollo) senza ulteriori modifiche.

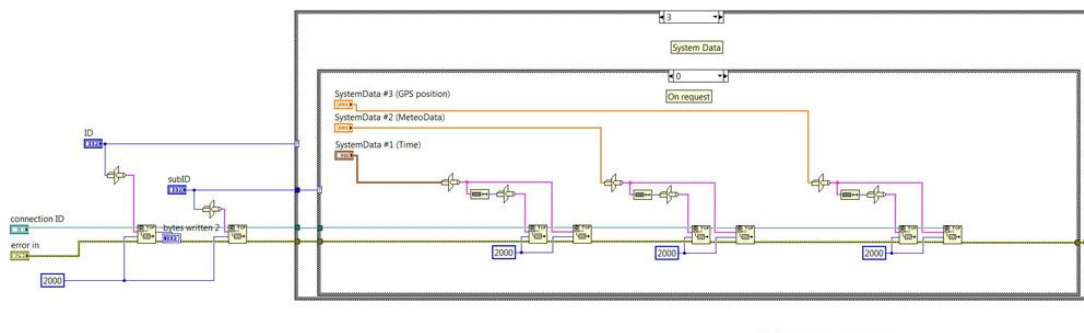



Figura 52 – VI di trasmissione dati da TCP (*s_TCPwrite.vi*). La trasmissione di ogni blocco di dati è anticipata dalla lunghezza di tale. Dal lato trasmissione *Typecast* è semplice poiché occorre soltanto portare il dato in stringa.

Vi sono tuttavia altri punti problematici tra cui la sincronizzazione dei vari cicli di trasmissione, poiché i dati da comunicare sono diversi e vanno trasmessi a frequenze differenti. Inizialmente si è inserito un controllo che inviasse la richiesta di trasmissione ogni qualvolta un tipo di dato richiedesse l’aggiornamento (on change, on request o temporizzato) ma si è riscontrato un problema di simultaneità di trasmissione. Infatti è possibile che un dato richiedesse la trasmissione mentre era ancora in processo un'altra. Così facendo la trasmissione in atto continuava il proprio iter mentre la nuova decadeva. Il problema è stato superato introducendo per ogni DP una coda, ovvero un *RT FIFO cluster* (First In First Out) in cui la richiesta di trasmissione è inserita in una sequenza di coppie (contenenti ID e subID del dato da inviare) che vengono processate dal ciclo di trasmissione in ordine di inserimento. In tal modo si ottiene di non perdere alcuna comunicazione ma si ammette un possibile ritardo nella comunicazione in caso in cui due o più richieste convergano nello stesso istante. In tal modo è necessario creare alcuni cicli paralleli a diversa temporizzazione per ogni frequenza di aggiornamento che vadano ad inserire il dato in coda. E’ necessario prestare attenzione a non sovraccaricare la coda, ad esempio

adottando qualche accorgimento, come l'inserimento di una richiesta in coda solo se la coppia (ID+subID) non è già presente[17]. Prima di inviare il dato, una parte del codice controlla che la coda non sia vuota, ed in tal caso usa l'ID riservato 50. In caso vi sia qualche elemento in coda, questo va ad aggiornare i corrispondenti dati.

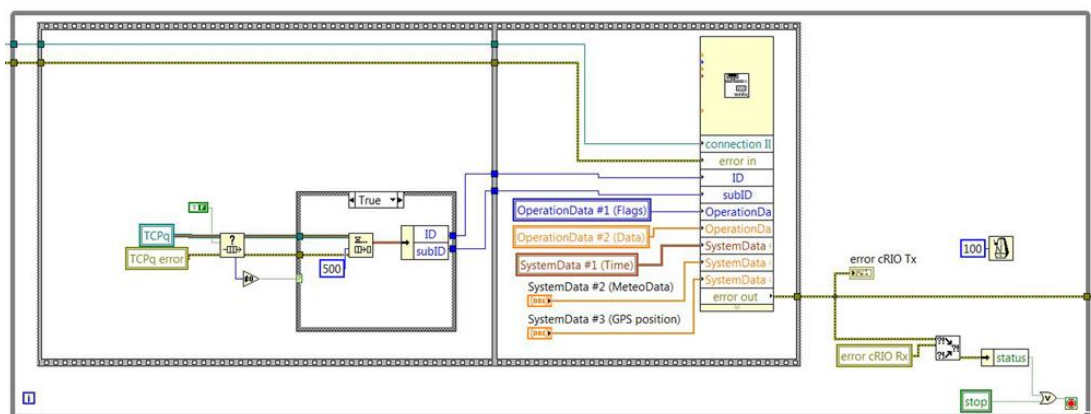



Figura 53 – Stralcio di codice che riporta la routine con cui si richiama la subVI di trasmissione. Nel primo frame della struttura sequenziale si controlla se vi siano dati da scrivere, cioè se sia presente una richiesta all'interno della coda *TCPq*. Nel secondo frame si procede alla trasmissione comunicando tutte le variabili necessarie alla subVI di trasmissione. Nel caso di coda vuota si entra nel caso True che setta ID e subID pari a 50. La presenza di un controllo complesso sugli errori e l'inserimento di un timeout sul ciclo di attesa permettono di evitare, in fase di chiusura della VI, che il ciclo entri in un loop infinito impedendo lo spegnimento.

4.4. STABILITA' DELLA CONNESSIONE

Tutto il ciclo di comunicazione TCP è racchiuso all'interno di un *ciclo While* temporizzato a 100 ms. All'interno è presente una struttura sequenziale. Il primo frame è utilizzato per l'inizializzazione degli indicatori di errore in ricezione e trasmissione *error Rx* ed *error Tx*. Nel secondo frame si procede all'apertura della connessione che avviene in maniera differente per cRIO e PC client.

Infatti il PC si connette ad un indirizzo IP stabilito dal server tramite la porta 5902 utilizzando il comando *TCP Open connection*. Per la cRIO l'apertura della connessione da lato server tramite il comando *TCP Listen* richiede soltanto il numero della porta. Questo comando apre una connessione verso qualunque dispositivo sia connesso sulla stessa porta.

[17] Questo controllo sarebbe da effettuare soltanto nel caso in cui il dato sia ad alta frequenza di aggiornamento. Per basse frequenze di aggiornamento, maggiori ai 300 ms, è lecito ritenere che la richiesta precedente sia già stata processata all'arrivo di una nuova.

In entrambi i casi si accede ad un *case* attivato dall'errore in uscita al comando di apertura della connessione. Nel caso *Error* avviene la reinizializzazione degli errori di trasmissione. Nel caso *No Error*, invece, si trova tutta la parte di comunicazione bidirezionale descritta dei capitoli 4.2 e 4.3.

Un aspetto importante per la stabilità della connessione è prevedere che i cicli interni al blocco TCP vengano fermati qualora si verifichi un errore che indichi perdita di connessione in trasmissione o più tipicamente in ricezione (comunemente *timout*, codice 56; *connection refused*, codice 63; *connection closed*, codice 66) o nel caso sia stato premuto il tasto di uscita. In questo modo si di può giungere alla chiusura della connessione nel caso la VI venga interrotta, o comunque all'uscita dalla struttura. L'uscita dai vari cicli richiede di unire i segnali d'errore provenienti da ognuno di essi. Tramite variabili locali *error Tx* ed *error Rx* si analizza lo stato della connessione e al più si giunge all'uscita dei vari cicli nel caso in cui sia presente un errore o sia stato premuto il pulsante di interruzione. Questo è realizzato grazie alla funzione *Merge error* che unisce gli errori del quale si ricava infine lo stato da *Unbundle by name*. Lo stato d'errore posto in somma logica con il segnale che richiede l'interruzione della VI, porta all'uscita del ciclo.

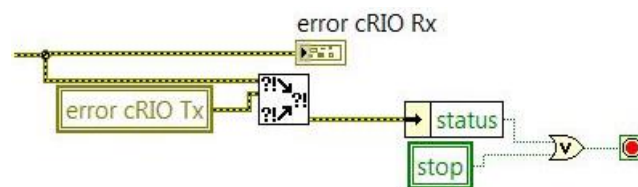



Figura 54 – Generazione del segnale di uscita dai cicli di ricezione e trasmissione TCP.

All'uscita da entrambi i cicli si accede ad un case in cui viene chiusa la connessione nel caso sia ancora aperta. Il case è attivato dal segnale in uscita dal comando *NotAPath* posto sull'ID di connessione. Questo permette di non generare errori e giungere alla chiusura di connessione, realizzata con l'istruzione *TCP Close Connection*.

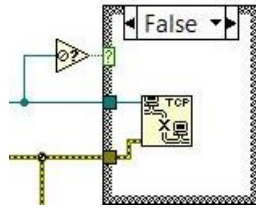



Figura 55 – Chiusura della connessione TCP.

Se la VI rimane attiva, avviene una nuova iterazione nel quale il client tenta la connessione fino al timeout al temine del quale si ferma ed incomincia con l'iterazione successiva. Grazie a questi accorgimenti la connessione è resa stabile a successive riconnessioni senza richiedere il riavvio del sistema, né della VI stessa. In particolare la connessione è stata testata ed risulta stabile anche in caso di disconnessione fisica del cavo di rete[18].

[18] La connessione TCP non risulta stabile se la VI operante su cRIO viene interrotta durante l'esecuzione dell'applicazione client. In questo caso sul client rimane aperta la porta di comunicazione, ed il client non riesce a riconnettersi a meno di un riavvio dell'intero software di LabVIEW. Questo caso è opportuno non includerlo negli eventi da gestire, poiché la VI su cRIO non deve mai essere interrotta.

Capitolo Quinto

CONTROLLO DEL SISTEMA

Gli azionamenti Siemens sono stati impostati per il controllo in velocità. Ciò vuol dire che il setpoint richiesto dall'utente - qualsiasi esso sia - deve essere elaborato fino ad ottenere un setpoint di velocità. Il controllo è svolto dal modulo RT e si divide in tre casi possibili di modalità:

- Manuale in velocità;
- Manuale in posizione;
- Automatico da polinomiale.

Il caso più semplice è ovviamente quello del controllo in velocità poiché in tal caso è l'utente stesso ad impostare il setpoint di velocità. Nel caso del controllo in posizione e del controllo automatico invece è necessario un sistema retroazionato che faccia raggiungere la posizione finale dell'asse tramite l'impostazione di una velocità. In tutti i casi è comunque necessario ottenere un setpoint di velocità, che viene poi filtrato fino ad ottenere un valore di velocità accettabile e pronto per essere trasmesso al motore.

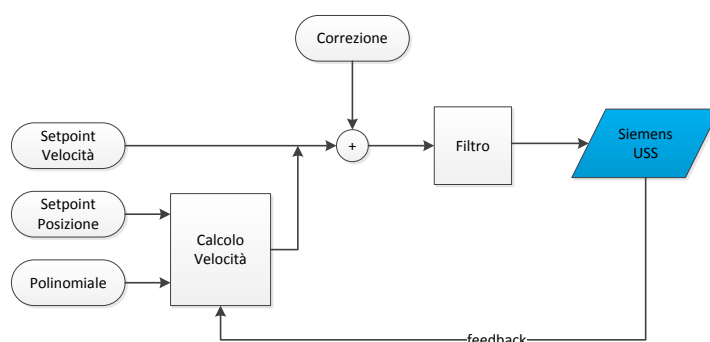



Figura 56 – Schema di impostazione del setpoint di velocità partendo da condizioni iniziali differenti. Da notare la presenza di una correzione che può rendersi necessaria per svariati motivi (nel caso più generale si ha un offset costante nullo). La presenza di un filtro permette di poter attenuare il setpoint introducendo una rampa, così da avere variazioni di velocità contenute in un intervallo accettabile sia da motori che dalla meccanica del sistema senza danneggiarsi.

Dal momento in cui si ottiene un setpoint di velocità richiesto dal sistema, l'algoritmo prosegue comune per tutti i casi – sia velocità, posizione o automatico.

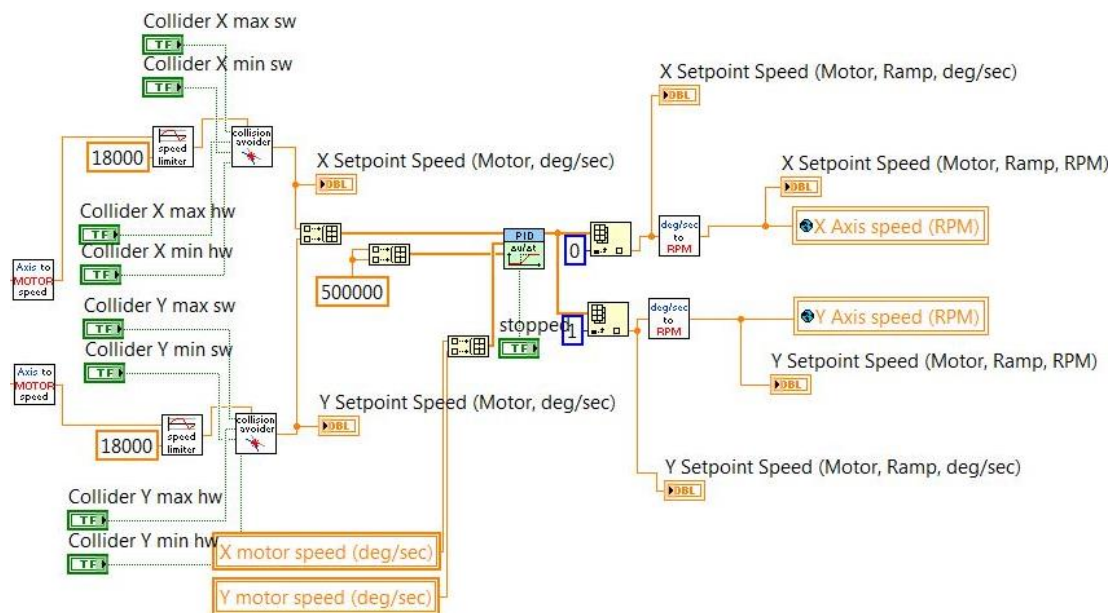



Figura 57 – Parte comune del codice di controllo del sistema. Questo algoritmo si sviluppa dall'ottenimento del setpoint di velocità fino al setting del setpoint corretto e filtrato agli azionamenti. Nella prima parte il blocco *sR_CollisionAvoider.vi* serve per limitare l'escursione massima dell'antenna. Il successivo *PID Output Rate Limiter* è utilizzato per la generazione di una rampa ed evitare accelerazioni troppo repentine. Le velocità così ottenute per ogni asse vengono impostate agli azionamenti tramite variabili globali *Axis Speed (RPM).* A questo componente occorre fornire i dati in ingresso sotto forma di array, e vengono dunque assemblati per i due assi con la funzione *Build Array*.

Tutta la parte di algoritmo precedente all'ottenimento del setpoint di velocità è differente per i vari casi ed è effettuata quindi in diversi case attivati da controlli sulle variabili del sistema. In particolare, si esegue un primo controllo sull'opzione *Manuale/Automatico* che introduce ad un *case*. Mentre nel caso vero - cioè di movimentazione automatica - vi è l'unica opzione del controllo in polinomiale che viene quindi attuata. In caso manuale si effettua un secondo controllo su *Velocità/Posizione*. Utilizzando nuovamente una struttura *case* si può accedere alle due modalità manuali. Per tutti e tre i casi si fornisce all'uscita del *case* più esterno un unico output di setpoint in velocità per ognuno dei due assi. In tal modo si svincola il filtraggio dei dati dal metodo con cui avviene il tracking o il posizionamento.

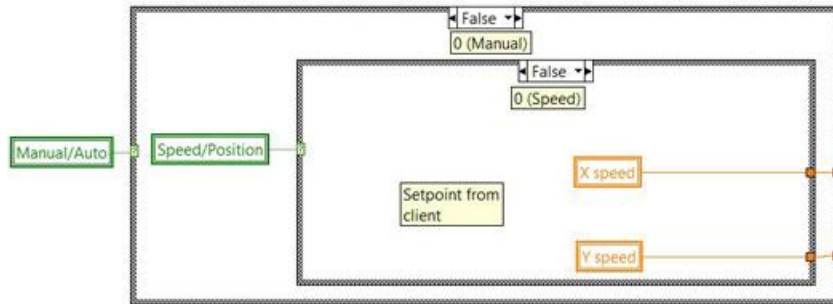



Figura 58 – *Case* annidati per il calcolo dei setpoint di velocità nelle varie modalità.

5.0.1. Correzione del setpoint di velocità

Nel corso di progettazione si è tenuto conto di un possibile errore del segnale dovuto a offset, disturbi o inconvenienti di trasmissione verso gli azionamenti. Per questo si è inserito un blocco di correzione del setpoint di velocità prima che questo sia inviato all'azionamento. L'algoritmo di controllo è un semplice sommatore che aggiunge al segnale richiesto una costante nulla, ma il fatto di averlo già previsto lascia posto a possibili correzioni in caso in fase di sperimentazione ci si accorgesse di avere un errore. La costante del sommatore può anche essere sostituita da un algoritmo di controllo più complesso ed elaborato, magari chiuso in ciclo di retroazione tra setpoint di velocità e comando motore.

5.0.2. Controllore di rampa

Per la movimentazione del sistema occorre tenere in considerazione che il corpo da ruotare non è costituito dal solo motore. Infatti il rotore è collegato ad una serie di riduttori di giri, che a loro volta movimentano l'asse dell'antenna. Considerando dunque che l'antenna è una parabola di circa 3 metri di diametro, questa opporrà una discreta inerzia ad accelerazioni e decelerazioni. Questa inerzia del sistema fisico porta a dover creare un filtro di limitazione di accelerazione, cioè di limitare la pendenza della retta tempo-velocità, evitando variazioni di velocità troppo brusche che porterebbero il motore a bloccarsi. Con una serie di prove sperimentali senza carico sull'asse si ottiene che l'aumento massimo di accelerazione in fase di test non deve superare i 500000 deg/min^2 all'asse del motore (tale valore andrà modificato in presenza dell'inerzia dell'antenna).

Per l'implementazione di un filtro che esegua tale limitazione è necessario ricorrere alla funzione integrata *PID Output Rate Limiter* che fornisce in uscita un setpoint di

velocità con una variazione massima impostabile. Questo componente richiede come ingressi un array del setpoint, il massimo rate di accelerazione e i valori di velocità attuali. Il vettore dei setpoint dei due assi è ottenuto costruendo un array (tramite funzione *Build Array*) con i valori calcolati, già convertiti in deg/sec sull'asse del motore e limitati alla velocità massima operativa del motore stesso (3000 RPM sull'asse motore, cioè 18000 deg/sec). Il vettore dei rate massimi di accelerazione è ottenuto tramite la funzione *Build Array* usando per entrambi gli assi la stessa costante. Infine, il vettore delle velocità attuali deriva dalla lettura dell'encoder motore. Al blocco viene fornito anche un ingresso che permette di resettare il valori iniziali di rampa. A questo è collegata la variabile booleana *stopped* che è vera solo nel primo ciclo successivo al riavvio della movimentazione. In questo caso, infatti, i motori sono fermi e si suppone che la rampa riporti a 0 il valore iniziale di velocità rilevata. In uscita dal *PID Output Rate Limiter* si ottiene un array i cui due elementi sono i setpoint di velocità limitati che vengono estratti con la funzione *Index Array* specificandone l'indice (che in questo caso coincide con l'ADR del motore corrispondente). I valori di velocità ottenuti sono calcolati in deg/sec e vengono convertiti in RPM prima di essere importati nella subVI *SiemensUSS.vi* tramite variabile globale *Axis Speed (RPM)*.

5.0.3. Contatti di fine corsa

Il sistema meccanico di ALMATracker presenta dei limiti su entrambi gli angoli (si veda il **capitolo 2.5**). Per questo è necessario prevedere dei comandi di fine corsa che inibiscano la movimentazione del sistema qualora venga richiesto di spostarsi in una posizione esterna al range di valori accettabili. I comandi di fine corsa hanno lo scopo di limitare il puntamento del sistema alle sole posizioni globalmente accettabili, tenendo conto della possibilità di interferenza tra i componenti meccanici. La subVI denominata *sR_CollisionAvoider.vi* si occupa di limitare movimenti agendo sui setpoint. Questa subVI, duplicata identicamente per i due assi, riceve cinque ingressi: setpoint richiesto, indicatori di collisione *max* e *min* da software e indicatori di collisione hardware. L'unica uscita della funzione è il setpoint corretto. Il blocco in questione viene interposto tra il setpoint richiesto e il controllore rampa. In questo modo è possibile ottenere una correzione in velocità (e quindi in posizione) senza alterare l'efficacia del limitatore di rampa. Gli indicatori di collisione si

dividono in due tipi: hardware e software. Gli indicatori hardware sono i microinterruttori già decritti nella parte hardware della meccanica (cap 2.5), mentre quelli software sono delle variabili booleane calcolate effettuando il confronto di superamento dei limiti massimi imposti dall'utente (cioè se l'angolo è compreso nel range di valori accettabili la variabile è falsa, altrimenti è vera). All'interno della subVI di fine corsa si effettua il controllo sugli indicatori di collisione. In particolare se la velocità è positiva (per velocità negative avviene tutto in maniera speculare) occorre verificare di non oltrepassare il limite di angolo massimo, effettuando il controllo sulla variabile *Collider max sw* posta in somma logica con *Collider max sw*, che a sua volta deriva da FPGA tramite un controllo in OR dei segnali dei due microinterruttori posti all'angolo massimo. Il blocco di fine corsa, se verifica che almeno uno degli indicatori di collisione è attivo, imposta a 0 la velocità fintanto che questa è positiva, lasciandola inalterata altrimenti (la decelerazione verrà poi limitata dal controllore di rampa a valle). Se invece nessuno di questi indicatori è attivo la velocità richiesta in uscita sarà lo stesso setpoint in ingresso. Tutte le verifiche vengono fatte per mezzo di comparatori logici il cui risultato permette l'accesso ad una serie di due case annidati.

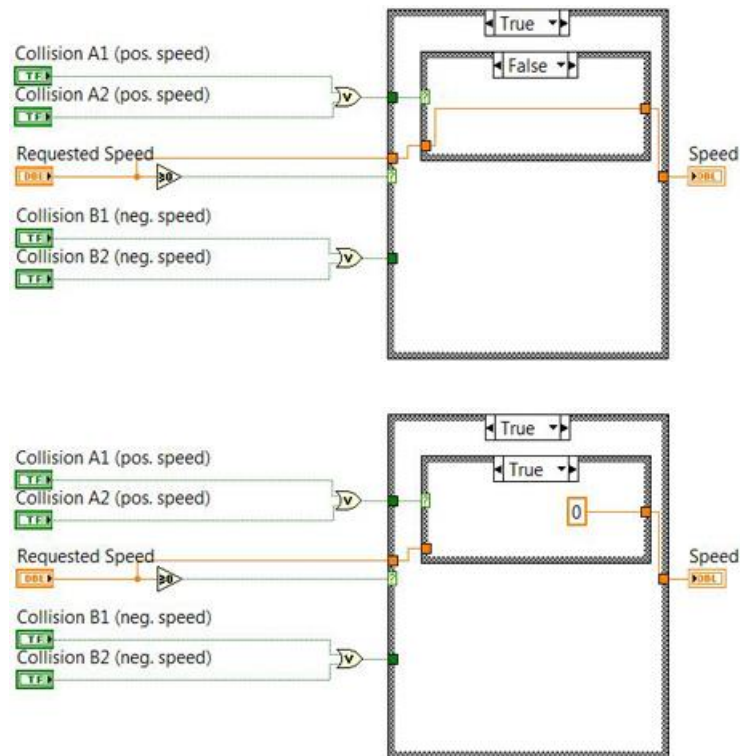



Figura 59 – *sR_CollisionAvoider.vi* presentato nei due casi – senza collisione nel primo screeshot, con collisione nel secondo – a velocità positive. In questa subVI i controller sono connessi secondo il seguente codice: la lettera maiuscola rappresenta la posizione dell'indicatore (**A** per max, **B** per min) mentre il numero rappresenta il tipo di indicatore (**1** software o **2** hardware).

5.1. SETPOINT DI VELOCITÀ

Il setpoint di velocità è impostato dall'utente tramite client. Il dato spedito via TCP giunge alla cRIO ed è pronto per essere elaborato. E' quindi sufficiente fornire come valore di output dalla struttura il valore letto da TCP tramite variabile locale, come mostrato in figura.

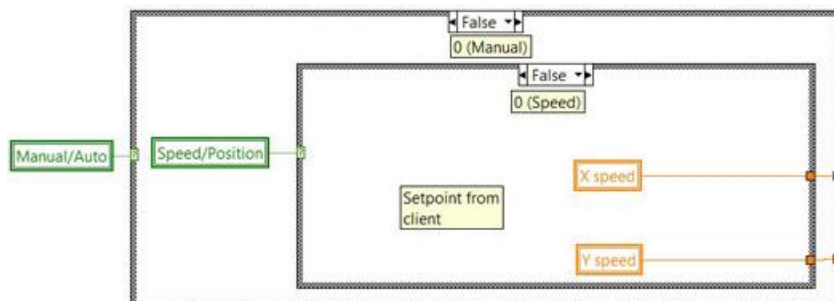



Figura 60 – Setting del setpoint di velocità fornendo il valore contenuto nella variabile locale, che un ciclo parallelo pone pari al valore impostato dall'utente via TCP.

5.2. *SETPOINT DI POSIZIONE*

Il controllo di posizione richiede un algoritmo più complicato del caso precedente. Innanzi tutto questo necessita obbligatoriamente di una retroazione per conoscere l'effettiva posizione del sistema, istante per istante. Si utilizza quindi una funzione che richiede come ingressi i setpoint di posizione e i valori degli angoli letti da encoder (entrambi derivanti da variabili locali) oltre ai valori minimi e massimi ammissibili[19] e riceve come output i setpoint di velocità, che saranno valori in uscita dalla struttura del caso in esame. La funzione che si occupa di eseguire l'elaborazione dei dati da un setpoint di posizione ad uno di velocità (*sR_PositionToSpeed.vi*) ricorre all'utilizzo di un controllore PID (Proporzionale - Integrativo - Derivativo).

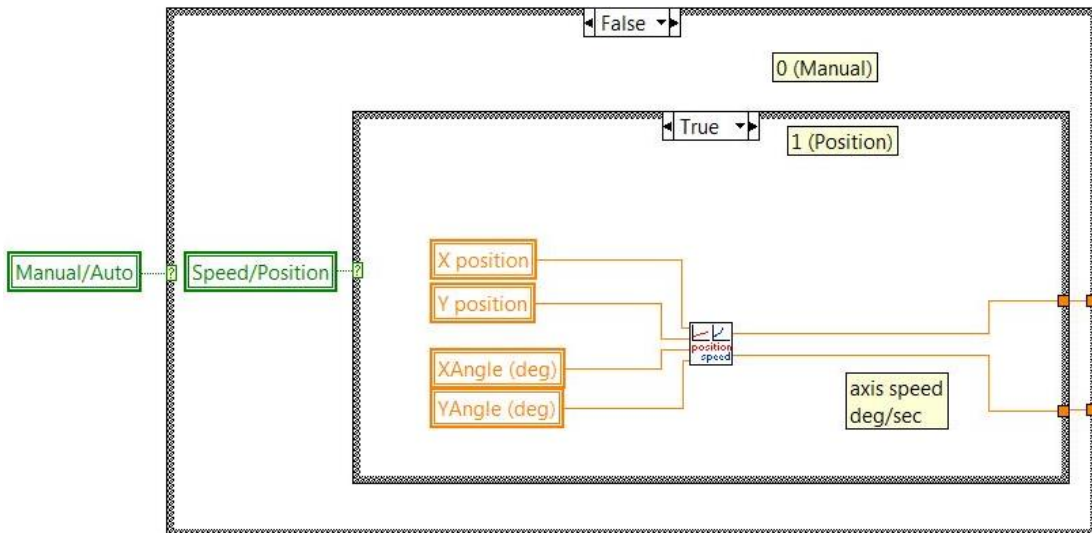



Figura 61 – Setting del setpoint di velocità a partire da uno di posizione.

Questo componente rappresenta un sistema in retroazione negativa, basato sull'inseguimento del setpoint da parte del sistema di controllo automatico. I sistemi PID sono largamente utilizzati in molti processi industriali poiché, modificando il comportamento di alcuni parametri, si riescono ad ottenere comportamenti differenti fino al raggiungimento di una risposta soddisfacente da parte del sistema complessivo di cui il PID controlla la dinamica. Il regolatore utilizzato ha una funzione di trasferimento standard

[19] I valori massimo di velocità è stato impostato a ±3000 RPM al motore, cioè ±9.18 deg/sec all'asse.

$$G(s) = K_p\left(1 + T_d s + \frac{1}{T_i s}\right)$$

La costante K_p si dice sensibilità proporzionale, T_d costante di tempo dell'azione derivativa e T_i costante di tempo dell'azione integrale. L'inverso della sensibilità proporzionale si dice banda proporzionale e corrisponde all'escursione della variabile controllata necessaria per provocare una variazione unitaria della variabile manipolabile, in assenza di azione derivativa ed integrativa[20]. Dunque, mentre il K_p fornisce una soglia per l'attivazione del controllore, il termine integrativo fa si che l'errore a regime sia nullo ovvero che il valore finale corrisponda alla richiesta. L'azione derivativa permette di variare la velocità di cambiamento dell'errore compensando la funzione per evitare che l'errore rimanga fisso per un certe periodo o aumenti oltremodo.

Per la regolazione dei valori delle guadagni del PID, fornite tramite un array composto da due *cluster* identici, si è ricorso ad un *tuning*[21] sperimentale del sistema. In particolare si è giunti ad impostare:

1. K_c (guadagno proporzionale) = 0.645;
2. T_i (tempo integrale) = 0;
3. T_d (tempo derivativo) = 0.014.

Questi valori sono risultati essere quelli che forniscono la risposta migliore in termini di tempo di ritardo e di assestamento, cioè del tempo con sui si arriva a regime.

La subVI *sR_PositionToSpeed.vi* è composta dal regolatore PID. Questo richiede in ingresso un range di valori accettabili di velocità (valori massimi e minimi), i setpoint di posizione, i valori attuali dei dati angolari e le costanti di guadagno. In uscita fornisce il setpoint di velocità. Tutti i dati scambiati con il blocco PID sono sotto forma di array, in cui ogni asse ha un indice (0 o 1). Tutti gli array vengono assemblati con la funzione *Build Array*, mentre l'estrazione avviene con un *Index Array*, indicando l'indice del valore da estrarre. I vettori di range e delle costanti di

[20] G. Marro, *Op. cit.* pag. 228.
[21] Il *tuning* è un metodo di regolazione dei parametri di un controllore PID attraverso la variazione dei valori fino ad ottenere il comportamento desiderato.

guadagno del PID sono formati unendo due cluster identici. Il vettore di setpoint è ottenuto unendo i valori derivanti da variabile locale dal blocco TCP, così come per il dato angolare letto che deriva direttamente dal blocco di lettura dell'angolo da encoder.

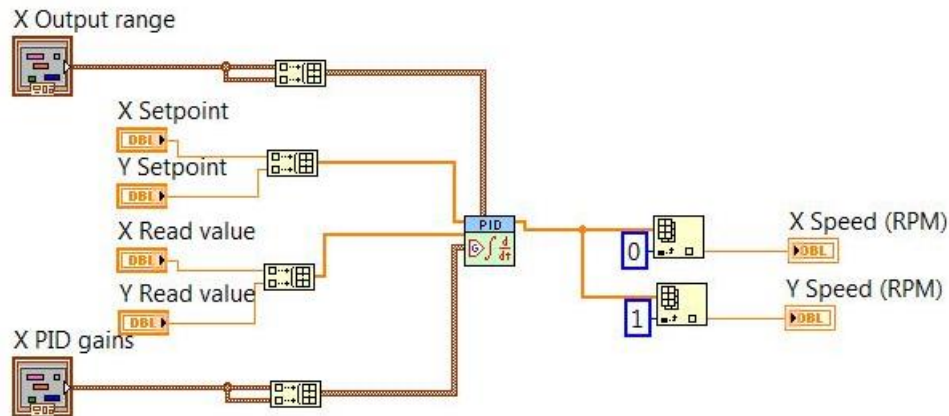



Figura 62 – *sR_PositionToSpeed.vi* per il calcolo della velocità partendo da un setpoint di posizione. Questa vi si avvale di un regolatore PID.

5.3. CONTROLLO SECONDO POLINOMIALE

Il controllo automatico consente di effettuare il tracking di un satellite inserendo da lato client le polinomiali delle posizioni istantanee sui due assi. Tali polinomiali, di grado non superiore al 40esimo sono della forma:

$$x(t_{JD})=a_0t^0+a_1t^1+\ldots+a_it^i+\ldots+a_{k-2}t^{k-2}+a_{k-1}t^{k-1}+a_kt^k$$

con $k\leq 40$, t_{JD}=Julian Date (tempo giuliano)

Per il calcolo di t_{JD} si ricorre alla conversione dal tempo "DateTime" del dispositivo cRIO a tempo giuliano utilizzando necessariamente lo stesso algoritmo del codice utilizzato per il calcolo della polinomiale. L'algoritmo utilizzato è dunque quello fornito da "The United States Naval Observatory[22]" e qui riportato:

```
int signJD=-1;

    if (((100*time_temp.year)+time_temp.month-190002.5)>0){
```

[22] http://www.usno.navy.mil/USNO/

signJD=1;

}else{

*if (((100*time_temp.year)+time_temp.month-190002.5)==0){*

signJD=0;

}}

*time_temp.JD=(367*time_temp.year) - floor((7*(time_temp.year+floor((time_temp.month+9)/12)))/4)+floor((275*time_temp.month)/9)+time_temp.day+1721013.5+((time_temp.min/1440.0)+(time_temp.hours/24.0)+(time_temp.sec/86400.0)+(time_temp.fraction_sec_double_precision/86400.0))-(0.5*signJD)+(0.5);*

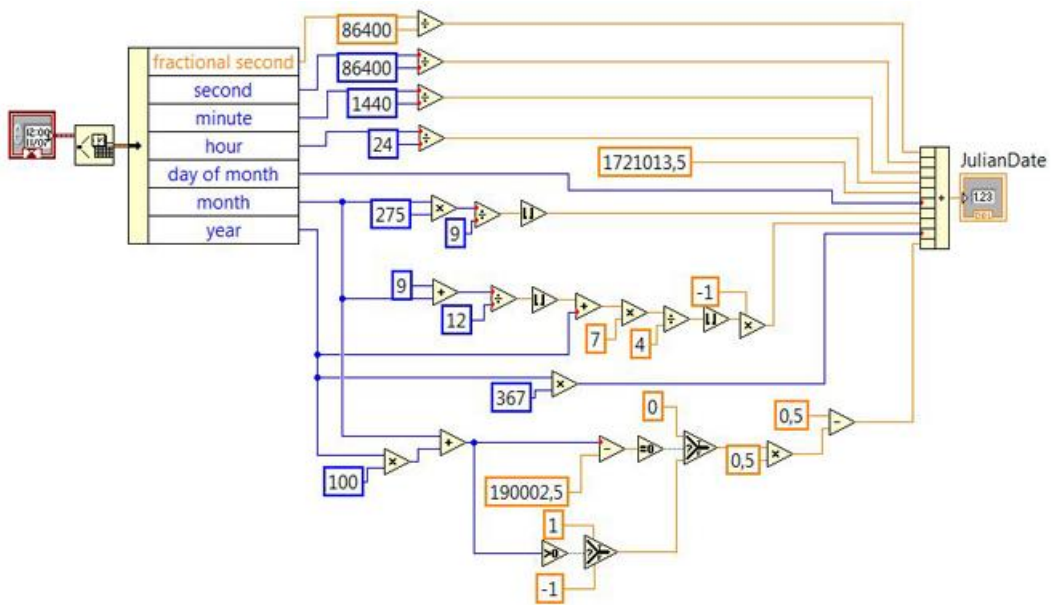



Figura 63 – *sR_JulianTime.vi* Algoritmo per il calcolo del Julian Date a partire dal tempo di sistema.

Poiché l'azionamento del sistema richiede un controllo in velocità risulta necessario passare da una polinomiale di posizione ad una di velocità. La subVI *DerivatePoly.vi* è stata programmata in modo da restituire in uscita una polinomiale derivata della polinomiale in ingresso, cioè per passare da x(t) a v(t). Per la derivazione si utilizza la formula:

$$a_i = \frac{a_{i+1}}{i+1}$$

con i=0:k

Dunque si elimina il primo termine a sinistra (a_0), tramite il comando *Delete From Array*, e si spostano tutti i termini a sinistra di un posto, moltiplicandoli per il loro vecchio indice che sarà pari al numero dell'iterazione, cioè *i+1*. La polinomiale ottenuta rappresenta v(t).

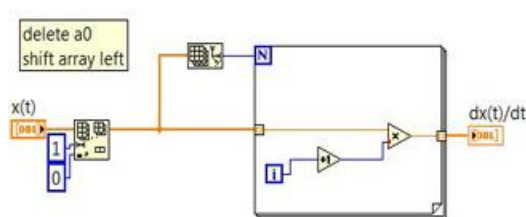



Figura 64 – Algoritmo di derivazione *sR_DerivatePoly.vi* .

La subVI *sR_SpeedFromPoly.vi* si occupa del calcolo della velocità e della posizione istantanee utilizzando le polinomiali e sostituendo il tempo calcolato in Tempo Giuliano. Questo si ottiene utilizzando un *ciclo For* di numero di iterazioni pari alla lunghezza della polinomiale. Due shift register contengono l'uno il valore del valore parziale della polinomiale calcolata, l'altro del tempo esponenziale. Il valore parziale della polinomiale si ottiene sommando al valore precedente del registro, inizializzato a 0, il prodotto tra il coefficiente della polinomiale ed il corrispondente tempo esponenziale. Quest'ultimo è ottenuto moltiplicando per ogni ciclo il valore del registro, inizializzato ad 1, al valore del tempo giuliano. In uscita dal ciclo il valore derivante dal registro del valore di polinomiale rappresenta la velocità istantanea o la posizione istantanea, a seconda della polinomiale in ingresso.

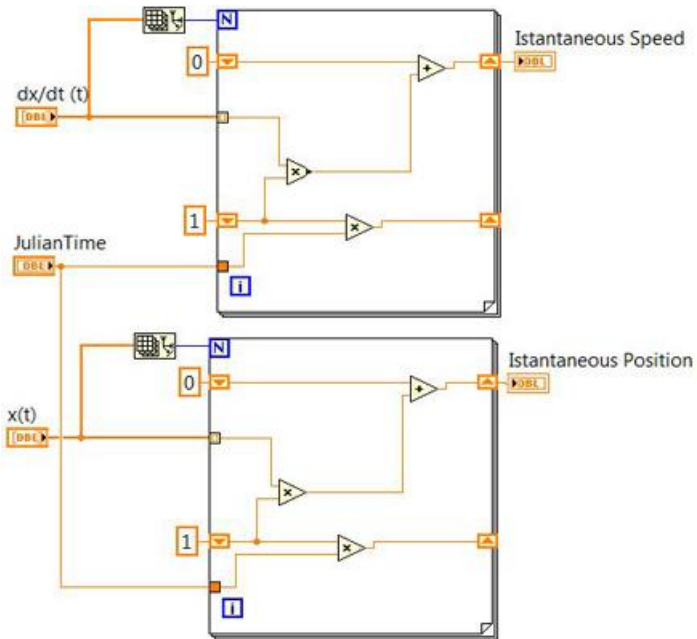



Figura 65 – *sR_SpeedFromPoly.vi* calcola i valori di velocità e posizione istantanee partendo dalle rispettive polinomiali e dal valore del tempo giuliano precedentemente calcolato.

Il valore della posizione è poi importato nella subVI *sR_PositionToSpeed.vi* precedentemente descritta nel controllo in posizione. Qui vengono dati gli ingressi delle posizioni attuali derivanti da encoder e quelli dei setpoint di posizione derivanti da polinomiale. A questo andrebbe aggiunto un secondo controllo di velocità che utilizza il feedback derivante dal calcolo svolto sul dato di encoder.

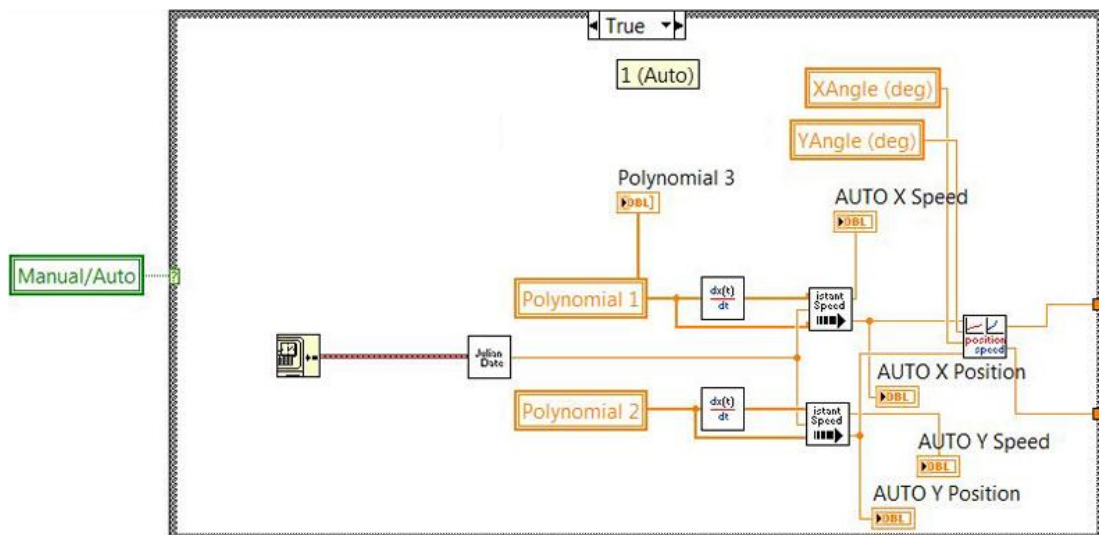



Figura 66 – Blocco di controllo automatico. E' possibile seguire il corso dei dati che dalle polinomiali passa ai blocchi di derivazione per il calcolo delle velocità, da qui ai blocchi di calcolo dei valori istantanei. Infine con il blocco di movimentazione in posizione si ottengono i setpoint di velocità.

Il tracking automatico rappresenta quindi una movimentazione in velocità ma ad inseguimento di posizione, con una velocità non stazionaria nel tempo. Questo richiede un algoritmo di controllo in catena chiusa, che utilizza anche i feedback di velocità rilevati dagli encoder. Il controllo automatico non è stato totalmente completato in questa sede. La tipologia di controllo per questa modalità è temporaneamente di tipo in posizione. La polinomiale di posizione fornisce il setpoint in funzione del tempo giuliano in ingresso ed il sistema provvede al raggiungimento dell’angolo.

Capitolo Sesto

CONCLUSIONI

Il software così come presentato è stato compilato per ottenere un eseguibile (*.exe) per il lato client, mentre su cRIO i bitfile generati sono stati caricati per permettere l’avvio automatico su accensione. Per poter operare sul sistema occorre impostare i dati di rete come di seguito riportati:

ALMATracker – *Impostazioni del sistema*	
Indirizzo IP cRIO	169.254.125.2
Subnet Mask	255.255.0.0
Porta comunicazione TCP	5902

Tabella 5 – Impostazioni di rete.

Il sistema così completato è stato sottoposto a fase di test per stabilirne la completa operatività e la stabilità in caso di eventi imprevisti. Al termine dei test di stabilità sono stati effettuate delle prove per le modalità di movimentazione manuale in velocità e in posizione. I risultati ottenuti sono riportati di seguito.

ALMATracker – *Prestazioni velocità*	
Velocità massima	±3˙000 RPM al motore (±18˙000 deg/sec) ±1.5306 deg/sec all’asse finale
Accelerazione massima	±500˙000 RPM/min al motore (±50˙000 deg/sec) ±25.5102 deg/sec all’asse finale
Tempo di accelerazione (0-V_{max})	< 2˙700 ms

Tabella 6 – Prestazioni di velocità.

Sebbene un qualunque sistema rotativo si comporti generalmente come un sistema del second’ordine, a causa dei parametri stabiliti al PID e della saturazione indotta

dal limitatore di accelerazione il sistema completo tende più ad un sistema del primo ordine, essendo nulla la sovraelongazione. In **Tabella 7** si riportano i valori generali di posizionamento comuni ad ogni valore del setpoint di posizione.

ALMATracker – *Prestazioni posizione I*	
Precisione di posizione	0.00549 deg
Accuratezza in posizionamento	< 0.01 deg
Massima sovraelongazione	0%
Istante di massima sovraelongazione	-

Tabella 7 – Prestazioni in posizione I.

La caratterizzazione del controllo in posizione di ALMATracker è descritta dai parametri caratteristici di un sistema in retroazione di ordine superiore al primo. Questi parametri forniscono informazioni sulla reattività del sistema e sulle risposte ai vari ingressi, riportate in **Tabella 8** e **Figura 68**.

ALMATracker – *Prestazioni posizione II*				
Setpoint	**90 deg**	**180 deg**	**270 deg**	**360 deg**
Tempo di ritardo [ms]	6·113	11·047	15·946	20·845
Tempo di salita [ms]	8·401	15·600	23·500	31·400
Tempo di assestamento$_{95\%}$ [ms]	12·212	20·447	29·346	38·545
Tempo di posizionamento [ms]	29·013	38·447	48·746	57·745

Tabella 8 – Prestazioni in posizione[23] II.

[23] Misurazioni riferite al sistema in partenza da fermo con setpoint di posizione. Tutte le misure riportate possono essere affette da un errore < 100 ms. Parametri come definiti in G. Marro, *op. cit.*, pagg. 63 e segg.

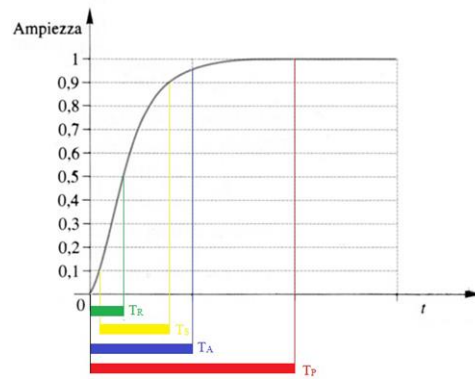



Figura 67 – Grafico di risposta di un sistema al gradino unitario, in cui sono evidenziati i parametri principali:
T_R è il tempo di ritardo, necessario al raggiungimento del 50% del valore finale.
T_S è il tempo di salita, necessario per passare dal 10% al 90% del valore finale.
T_A è il tempo di assestamento, necessario perché la risposta sia compresa in un intervallo che non si discosta dal valore finale più del ±5%.
T_P è il tempo di posizionamento, necessario al raggiungimento del valore finale.

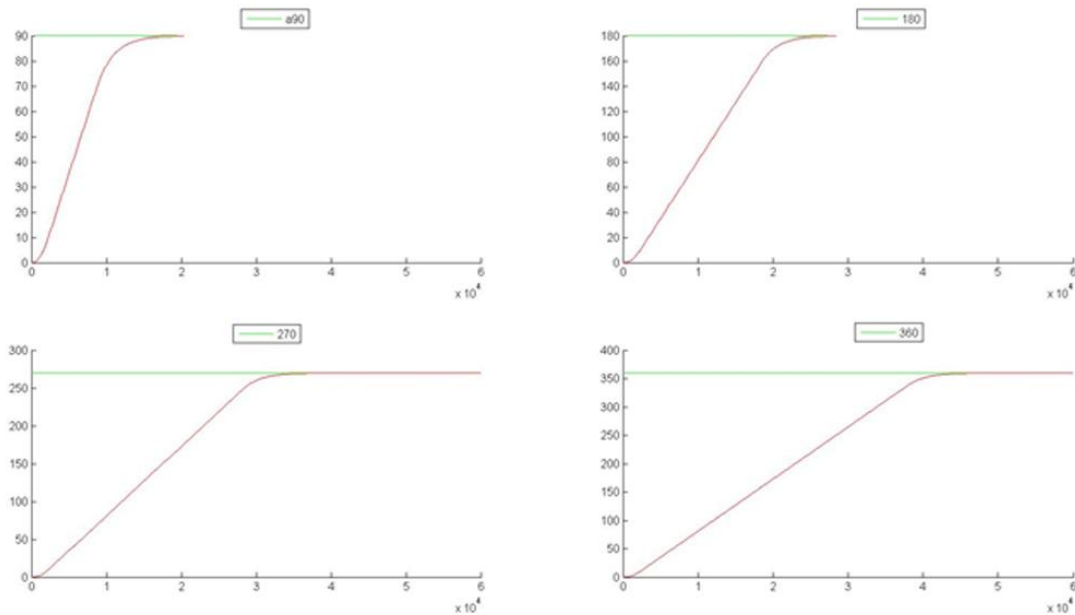



Figura 68 – Profili di posizione per vari setpoint (90, 180, 270 e 360 deg). In ascisse si riportano i tempi, in ms dall'avvio del sistema; in ordinate la posizione.

6.1 SVILUPPI FUTURI

Il software di ALMATracker sviluppato nel corso di questa tesi rappresenta la struttura principale della versione finale del programma. Il sistema, infatti, permette il completo controllo della movimentazione in posizione e velocità. La modalità automatica, invece, è stata sviluppata in maniera marginale, utilizzando una parte dell’algoritmo della movimentazione in catena chiusa con feedback in posizione. Si

ritiene che questo debba essere sostituito da un doppio feedback in posizione e velocità, al fine di un miglioramento dell'accuratezza e la reattività di posizionamento. Sempre nell'ambito della modalità automatica sarebbe opportuno includere il software di propagazione e predizione dell'orbita dello spacecraft all'interno del sistema ALMATracker, svincolando il programma da altri e riducendo il lavoro richiesto all'utente.

Altri aspetti da migliorare riguardano l'ampliamento del sistema hardware per la ricezione di più segnali di fine corsa degli assi, come descritto nel capitolo 3.1.3 (pag. 23). L'ampliamento hardware è richiesto anche per i dati meteo e GPS. In particolare i dati di posizionamento globale sono richiesti dal software di calcolo della polinomiale e sarebbe conveniente averli a disposizione.

Appendice

ALMATRACKER TRANSMISSION PROTOCOL (ATTP)

0. CONVENZIONI DATI

Per convenzione si definiscono le unità di misura utilizzate nello scambio dati interno al software ALMATracker.

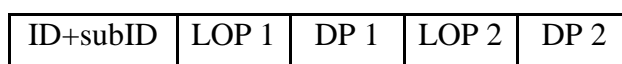
Misura	**Unità di misura**
Angoli	Deg
Velocità angolari	Deg/sec
Velocità lineari	Km/h
Temperature	°C
Pressioni	Pa
Tassi di umidità	Nessuno, espresso in percentuale (%)

1. CODIFICA DATI

La stringa di trasmissione TCP/IP è codificata secondo il seguente formato:

ID+subID	2 byte, riferimento nella sezione 1.1
LOP (Length Of Package)	4 byte, include la lunghezza dei pacchetti successive espreressa come intero a 32 bit (I32)
DP (Data Package)	Di lunghezza definita (LOP), contiene i dati riferimento alla sezione 1.3

Per esempio: una trasmissione di un tipo di dato (ID) che coinvolge due pacchetti[24] (DP 1, DP 2), dovrebbe essere spedita come segue.

ID+subID	LOP 1	DP 1	LOP 2	DP 2

[24] Differenti pacchetti di dati dovrebbero essere utilizzati per ogni tipo di dato.

1.1. ID e subID

L'ID di trasmissione designa la categoria di data che verranno inviati. Deve essere scelto dalla lista riportata di seguito, a seconda del dato che si intende trasmettere.

<table>
<tr><th>ID</th><th>subID</th><th>Data Category</th><th>Data Flow</th></tr>
<tr><td>0</td><td>0</td><td>Operation Data</td><td>cRIO to Client</td></tr>
<tr><td>1</td><td>0</td><td>Polynomial</td><td>Client to cRIO</td></tr>
<tr><td rowspan="4">2</td><td></td><td>AlmaTracker Controls</td><td rowspan="4">Client to cRIO</td></tr>
<tr><td>0</td><td>Controls</td></tr>
<tr><td>1</td><td>Setpoint</td></tr>
<tr><td>2</td><td>Time</td></tr>
<tr><td>3</td><td>0</td><td>System Data</td><td>cRIO to Client</td></tr>
<tr><td>50</td><td></td><td>Void case</td><td>any</td></tr>
</table>

1.2 Length Of Package

Questi quattro byte contengono la lunghezza (in byte) del DP successive. Ogni LOP si riferisce esclusivamente al solo DP che lo segue, non all'intera trasmissione.

1.3 Data Package

Coerentemente con l'ID scelto, il DP deve essere compilato con i dati come richiesto nel seguito.

ID 0 (Operation Data)

subID 0

Questo pacchetto è format da due DP differenti. DataPack#1 è un DP di 6 byte contenente flag con informazioni sia sul sistema, che su eventuali allarmi. DataPack#2 è un array di double di 48 byte contenente le informazioni di setpoint ricevuti ed impostati dal software agli azionamenti. Qui vi sono anche le informazioni angolari e di velocità derivanti dai vari feedback.

<u>DataPack #1 (Flags)</u>

Dati da Siemens USS

- X-Axis Control Flag;
- Y-Axis Control Flag;

- X-Axis Status Flag;
- Y-Axis Status Flag;

Dati impostazioni cRIO

- Start/Stop status;
- Manual/Auto status;
- X Ignition Fatal Error;
- Y Ignition Fatal Error;
- Posizion/Speed status;

Dati sengali fine corsa

- X max collision sw;
- X min collision sw;
- Y max collision sw;
- Y min collision sw;
- X max collision hw;
- X min collision hw;
- Y max collision hw;
- Y min collision hw;
- *Vuoto [spazi vuoti disponibili: 3 bit]*.

<u>DataPack #2 (Data)</u>

Misurati da encoder

- X Angle;
- Y Angle;
- X Speed;
- Y Speed;

Dati da Siemens USS

- X Speed;
- Y Speed;

cRIO Setpoints

- X Axis Setpoint;
- Y Axis Setpoint;
- X Speed AUTO;
- Y Speed AUTO;
- X Position AUTO;
- Y Position AUTO.

ID 1 Polynomial

subID 0

Costituito da due array di double (41 termini ciascuno). Contiene i coefficienti delle polinomiali di posizione di 40esimo grado (k=40, $x(t)=a_0t^0+a_1t^1+\ldots+a_it^i+\ldots+a_{k-2}t^{k-2}+a_{k-1}t^{k-1}+a_kt^k$). I coefficienti sono ordinati per grado crescente (da a_0 a a_{k-1}). Il tempo è per convenzione il Tempo Giuliano (Julian Date) calcolato con l'algoritmo fornito da *The United States Naval Observatory*[25].

ID 2 AlmaTracker Controls

In questo campo vi sono tutti I controlli che è possibile abilitare dall'interfaccia utente del client. i subID sono utilizzati per dividere DP a differenti tempi di aggiornamento.

subID 0 (Controls)

DataPack#1 è un DP, della dimensione di 1 byte, contenente i controlli principali impostati dall'utente.

DataPack #1

- Stop/Start *(Stop=0)*;
- Manual/Auto *(Manual=0)*;
- Speed/Position *(Speed=0)*;
- Parking *(0 inactive)*;
- Synchronize data/time (00 or 11 nessuna sincronizzazione, 01 da PC, 10 da NTP server);
- *Vuoto [spazi vuoti disponibili: 2 bit]*.

subID 1 (Setpoint)

DataPack#1 è una stringa di double di 32 byte contente i setpoint per la posizione e la velocità dei due assi e i valori di fondo scala impostati dall'utente.

DataPack #1

Setpoint sistema

- X position;
- Y position;
- X speed;

[25] http://www.usno.navy.mil/USNO/

- Y speed;

Fine corsa hardware

- X Max Range;
- X Min Range;
- Y Max Range;
- Y Min Range.

subID 2 (Time)

DataPack#1 è un cluster di interi U16 contenente l'orario con cui la cRIO deve essere sincronizzata (se richiesto).

<u>DataPack #1(Orario)</u>

- Hh;
- Min;
- Sec;
- M;
- D;
- Y.

ID 3 System Data

subID 0

Questo caso presenta tre differenti DP. DataPack#1 è un cluster di interi U16 contenente l'orario, DataPack#2 un array di 5 double (40 byte) , DataPack#3 an array of 3 double (24 byte).

<u>DataPack#1 (Orario)</u>

- Hh;
- Min;
- Sec;
- M;
- D;
- Y.

<u>DataPack#2 (Dati meteo)</u>

- Temperature;
- Wind direction;
- Wind intensity;
- Pressure;

- Humidity.

DataPack#3 (Dati GPS)

- Latitude;
- Longitude;
- Altitude.

CENNI SU LABVIEW

Si riportano di seguito alcune nozioni comuni nell'utilizzo di LabVIEW, per aiutare il lettore nella comprensione dell'elaborato.

Variabili:

Nome	Dimensione	Tipo dato
Boolean	1 bit	True/False
U32	32 bit	Numeri interi senza segno [0; 4294967295]
I32	32 bit	Numeri interi con segno [-32768; +32767]
DBL (double)	8 bytes	Numeri reali con segno

Definizioni:

Nome	Definizione
Front panel	Interfaccia grafica di una VI di LabVIEW.
Block diagram	Interfaccia di programmazione di una VI di LabVIEW.
Ciclo While	Ciclo che esegue una data istruzione fino al verificarsi di una condizione che ne comporta il termine.
Ciclo For	Ciclo che esegue una data istruzione per un numero prefissato di iterazioni.
Shift register	Variabile che all'interno di un ciclo permette di memorizzare il valore tra due iterazioni successive.
Select?	Comando che permette di selezionare un valore tra due possibilità, a seconda del valore di una variabile di controllo di tipo Boolean.
Typecast	Comando per il casting di un valore da un tipo ad un altro, cioè del cambiamento di rappresentazione di una variabile o una costante.
Build Array	Permette di costruire un array inserendo gli elementi uno ad

	uno.
Index Array	Permette di estrapolare un elemento da un array specificandone l’indice di posizione.
Algoritmo/Routine	Insieme di istruzioni che realizzano un certo comportamento.
VI	Virtual Instrument, interfaccia del software LabVIEW.
Canale DIO	Canale *Digital Input Output* rappresenta un canale digitale per la comunicazione dei dati in uno dei due sensi.

SUBVI SVILUPPATE

In questa parte di appendice sono inserite le varie subVI minori sviluppate nell'ambito del progetto ALMATracker ma non descritte all'interno dell'elaborato.

Ogni subVI si trova all'interno del proprio dispositivo (PC o cRIO) nell'apposita cartella virtuale *SubVI*. Anche i file salvati su disco riportano tutte le subVI nella cartella *subVIs*, che le differenzia dalle VI principali salvate in *mainVIs*. Le subVI sono inoltre dotate di un nome codificato del tipo:

sX_Yyyyyy.vi

dove le sigle rappresentano:

- *"s"* la dicitura "subVI";
- *"X"* una lettera per indicare il modulo che ne fa il principale utilizzo: "P"(PC), "R"(cRIO) o "F"(FPGA);
- *"Yyyyyyy"* è una descrizione del task della VI.

BCC *sR_BCCcalculator.vi*

Questa VI serve per il calcolo del carattere di controllo di errore della comunicazione con gli azionamenti. In particolare si utilizza l'array proveniente dall'azionamento calcolandone il carattere BCC come somma logica esclusiva (XOR) tra ogni carattere della stringa ed il BCC calcolato nell'iterazione precedente. Questo si ottiene eseguendo un *ciclo For* di tante iterazione quanti sono i bit dell'array iniziale, letti tramite il comando *Array Size*. All'interno del ciclo si esegue lo XOR tra il carattere i+1 dell'array ed il precedente carattere calcolato, contenuto in un registro. All'uscita del ciclo il carattere contenuto nel registro rappresenta il BCC completo e viene concatenato con l'array di partenza tramite il comando *Build Array*.

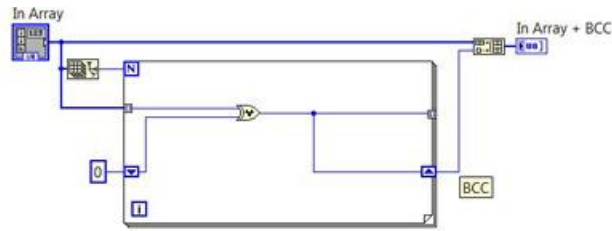



Figura 69 – *sR_BCCcalculator.vi*

s_OperationDataFlags.vi

L’utilità della presente VI è quella di poter ricavare dal pacchetto dati *OperationData #1 (Flags)* tutto il set di indicatori. In particolare la VI si avvale della funzione *Number To Boolean Array* per ricavare dal numero intero il vettore che poi è diviso tramite *Array Subset*, specificando il byte di partenza ed il numero di byte da considerare. Per alcuni sottoarray inoltre si considerano alcuni bit separati utilizzando il comando *Index Array* e specificando quali bit isolare.

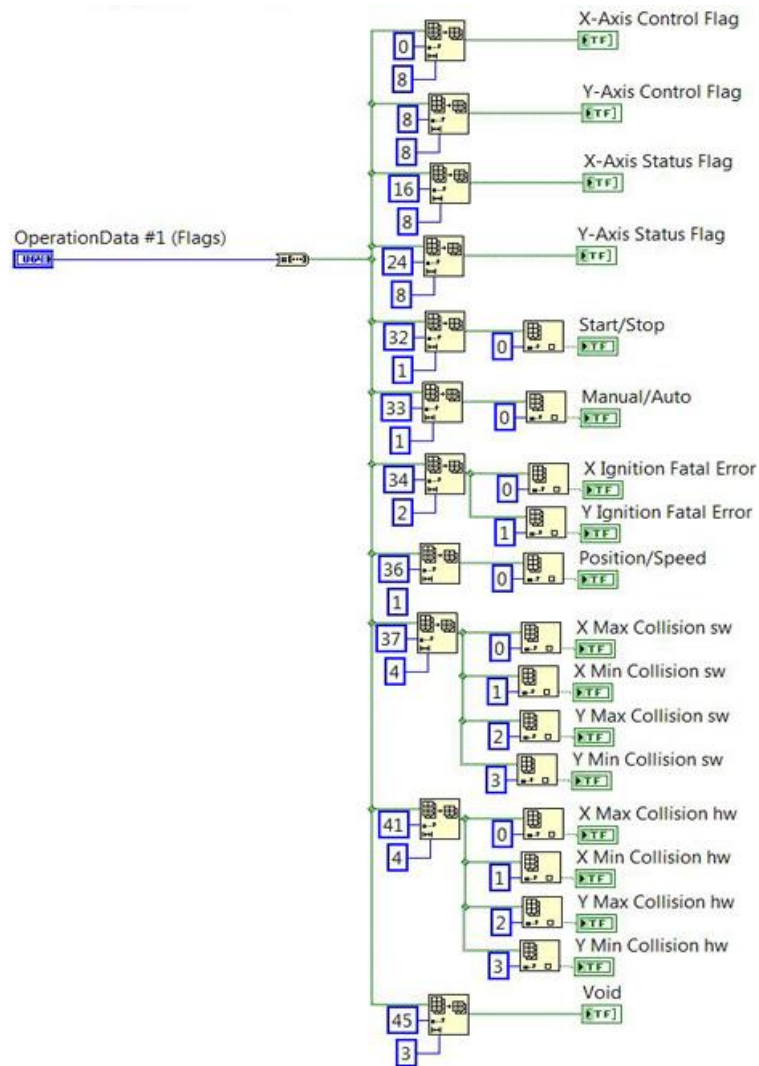



Figura 70 – *s_OperationDataFlags.vi*

sR_AlmaTrackerControls.vi

La VI in questione è stata creata per ottenere i flags contenuti nell'intero U64 del DP *Controls* di *AlmatTrackerControls*. Tramite la funzione *Number To Boolean Array* l'intero viene convertito in un vettore di booleani. A sua volta il vettore viene diviso utilizzando la funzione *Index Array* per ricavare, con indici differenti, i corrispondenti flag.

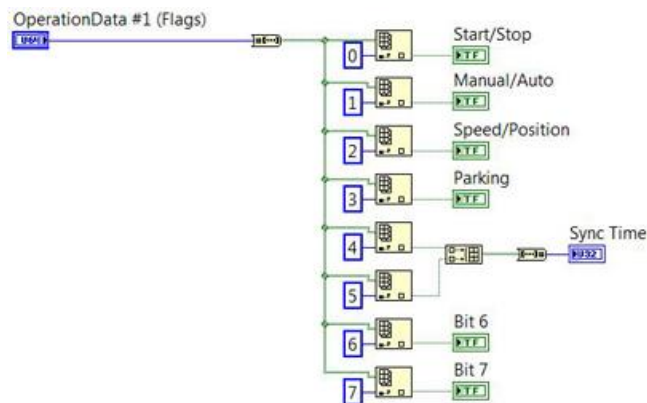



Figura 71 – *sR_AlmaTrackerControls.vi*

s_ArrayToBoolean.vi

Tramite una serie di *Index Array* permette di separare tutti gli elementi di array booleani fino ad 8 termini.

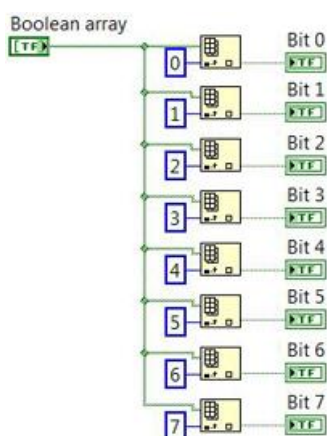



Figura 72 – *s_ArrayToBoolean.vi*

s_TimeBuilder.vi

Questa VI serve per l'assemblamento del dato di tempo nel formato richiesto da ATTP. In particolare si esegue un *Unboudle By Name* seguito da un *Boundle By Name*. In tal modo si crea un cluster con tutti e soli i valori del tempo utili al fine dell'applicazione. In particolare si escludono tutto ciò che è al di sotto del secondo. Il dato finale viene convertito in secondi tramite *Date/Time To Second*.

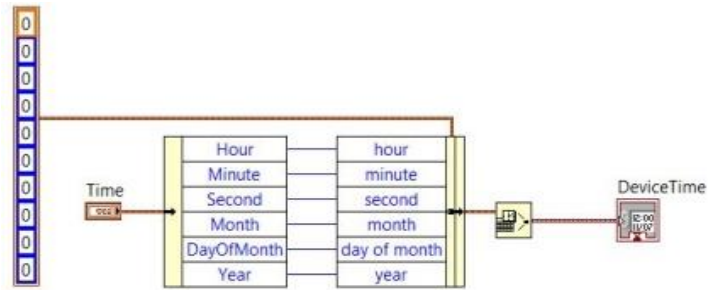



Figura 73 – *s_TimeBuilder.vi*

sR_TheresholdMotorSpeed.vi

Questa VI è un filtro che riporta la velocità richiesta all'interno di un range di valori. Infatti il valore assoluto di *Motor Speed* viene comparato con la velocità massima ammissibile tramite l'istruzione *Max & Min*, che fornisce in uscita il valore minimo tra i due. Questo valore moltiplicato per il segno di *Motor Speed* rappresenta la velocità impostabile all'azionamento più vicina alla richiesta. Questo comando equivale ad impostare il valore di velocità pari al massimo se la richiesta è maggiore di tale valore, o il setpoint richiesto altrimenti.

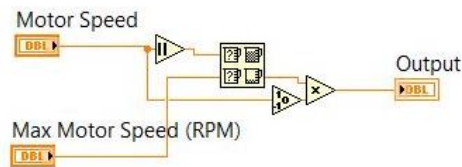



Figura 74 – *sR_TheresholdMotorSpeed.vi*

s_DegpersecToRpm.vi

Questa semplice VI utilizza una divisione per convertire una velocità in deg/sec in una espressa in RPM. Considerando che

$$1\ RPM = \frac{360\ deg}{1\ min} = \frac{360\ deg}{60\ sec} = 6\ deg/sec$$

Allora la velocità in RPM è pari a

$$\omega_{RPM} = \frac{\omega_{deg/sec}}{6}$$

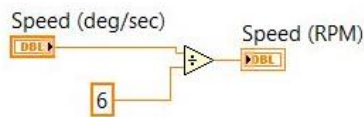



Figura 75 – *s_DegpersecToRpm.vi*

s_RpmToDegpersec.vi

Questa VI esegue la funzione inversa della precedente. La velocità in RMP viene convertita in deg/sec secondo la relazione $\omega_{deg/sec} = 6 * \omega_{RPM}$

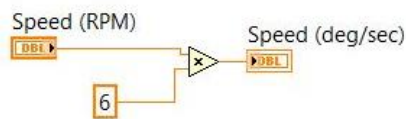



Figura 76 – *s_RpmToDegpersec.vi*

s_MotorToAxisSPEED.vi

La VI considerata è utilizzata per il calcolo della velocità sull'asse partendo dalla velocità del motore. I due riduttori 1:28 ed 1:70 posti in serie impongono di dividere la velocità per i coefficienti di riduzione.

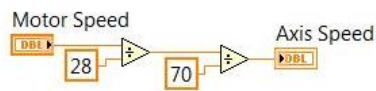



Figura 77 – *s_MotorToAxisSPEED.vi*

s_AxisToMotorSPEED.vi

Questa VI esegue la funzione inversa della precedente. Partendo dalla velocità all'asse viene calcolata la velocità al motore moltiplicando per i rapporti di riduzione.

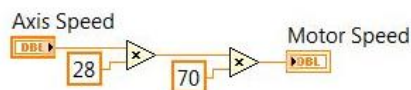



Figura 78 – *s_AxisToMotorSPEED.vi*

ALMATRACKER REPORT STS.M

```
%Almatracker report STS
clear all;
clc;
close all;
echo off;
data=fopen('ALMATracker_report_STS1.txt','r');
A = fscanf(data, '%g %g %g', [3 inf]);
A=A';
%matrici a 3 colonne per i 3 diversi tick 100.000 | 10.000 | 1.000
SP=zeros(8400,3);
ENC=zeros(8400,3);
intENC=zeros(8400,3);
for i=1:3
    for j=1:8400
        k=(i-1)*8400+j;
        SP(j,i)=A(k,1);
        ENC(j,i)=A(k,2);
        intENC(j,i)=A(k,3);
        if SP(j,i)==inf
            SP(j,i)=A(k+1,1);
        end
        if ENC(j,i)==inf
            ENC(j,i)=A(k+1,2);
        end
```

```
        if intENC(j,i)==inf
            intENC(j,i)=A(k+1,3);
        end
    end
end
%studio dei casi
%diviso per tick e sample
sp=zeros(700,1);
enc=zeros(700,3);
intenc=zeros(700,1);
%diviso per velocità
zsp=zeros(50,1);
zenc=zeros(50,3);
zintenc=zeros(50,1);
%init media
menc=zeros(1,3);
%matrice con valore di errore %: righe=sample colonne=vel
RMSE=zeros(14,12);
color=['-b','-k','-g','-r','-y','-c','-m',':b',':g',':c',':r',':m'];
%vettore di errore relativo
ERROR=zeros(12,2);
for c=1:3   %per i vari tick
    for k=0:11   %per ogni velocità
        %inizializzo errore relativo a 100%
        for i=1:700 %estrapolo dato utile per questa velocità
            sp(i,1)=SP(k*700+i,1);
```

```
        enc(i,:)=ENC(k*700+i,:);
        intenc(i,1)=intENC(k*700+i,1);
    end
    for i=0:13  %per ogni sample
        for j=1:50  %per ogni ciclo di lettura
            zsp(j,1)=sp(i*50+j,1);
            zenc(j,:)=enc(i*50+j,:);
            zintenc(j,1)=intenc(i*50+j,1);
        end
        %elaborazione e calcolo RMSE
        msp=mean(zsp(:,1));
        mintenc=mean(zintenc(:,1));
        menc(1,c)=mean(zenc(:,c));
        rmse=abs((msp-menc(1,c))/msp*100);   %rispetto al SP
        if ((msp==0)&&(menc(1,c)==0))
            rmse=0;
        end
        RMSE(i+1,k+1)=rmse;
        %errore sui dati fascia 1
        if (msp<=196)
            if (i==13)&&(c==1)
                fprintf('Speed= %4d\tTick= %6d\tSample= %4d\tERR= %6f\n', msp,
10^(3-c)*1000, 100+(i*100), rmse);
                ERROR(k+1,1)=menc(1,c);
                ERROR(k+1,2)=rmse;
            end
```

```
        end
        %errore sui dati fascia 2
        if (msp<=1960)&&(msp>196)
            if (i==4)&&(c==2)
                fprintf('Speed= %4d\tTick= %6d\tSample= %4d\tERR= %6f\n', msp,
10^(3-c)*1000, 100+(i*100), rmse);
                ERROR(k+1,1)=menc(1,c);
                ERROR(k+1,2)=rmse;
            end
        end
        %errore sui dati fascia 3
        if (msp<=9800)&&(msp>1960)
            if (i==1)&&(c==2)
                fprintf('Speed= %4d\tTick= %6d\tSample= %4d\tERR= %6f\n', msp,
10^(3-c)*1000, 100+(i*100), rmse);
                ERROR(k+1,1)=menc(1,c);
                ERROR(k+1,2)=rmse;
            end
        end
        %errore sui dati fascia 4
        if (msp>9800)
            if (i==0)&&(c==3)
                fprintf('Speed= %4d\tTick= %6d\tSample= %4d\tERR= %6f\n', msp,
10^(3-c)*1000, 100+(i*100), rmse);
                ERROR(k+1,1)=msp;
                ERROR(k+1,2)=rmse;
            end
```

```
            end
        end
        figure(c);
        hold on;
        for n=1:12
            plot(RMSE(:,n),color(1,(n-1)*2+1:(n-1)*2+1+1));
        end
        str = sprintf('Tick = %d',10^(3-c)*1000);
        title(str);
        grid;
        xlabel('Sample(x100)');
        ylabel('%RMSE');
legend('v=0','v=0.5','v=1','v=1.5','v=2','v=2.5','v=3','v=3.5','v=4','v=4.5','v=5','v=5.5');
        hold off;
    end
end
figure(4);
hold on;
plot(ERROR(:,1),ERROR(:,2),'+r');
title('Error in speed estimation');
xlabel('Motor speed');
ylabel('% RMSE');
legend('Step by step function');
grid;
hold off;
```

Riferimenti Bibliografici

A.A.V.V., *CompactRIO™ and LabVIEW™ Development Fundamentals – Course Manual*, National Instruments Corp., Hungary, 2008

G. Marro, *Controlli Automatici* - Quinta Edizione, Zanichelli, Bologna, 2006

Documenti online

Protocollo USS®:

http://cache.automation.siemens.com/dnl/DU0MjczAAAA_24178253_HB/uss_24178253_spec_76.pdf

Protocollo SSI:

http://www.tecnopower.es/media/html/CE/EA/09_ssi.pdf

http://www.todescato.com/Doc/Encoder/Tekel/Interfaccia%20SSI.pdf

Alcuni estratti di programma possono essere stati tratti da esempi e/o suggerimenti delle community di NI o LabVIEW Italia forum:

http://zone.ni.com/dzhp/app/main

http://www.ilvg.it/forum/viewforum.php?f=1

RINGRAZIAMENTI

Desidero innanzitutto ringraziare il Professore Paolo Tortora per avermi permesso di sviluppare questa tesi e per la disponibilità dimostrata.

Un doveroso ringraziamento va anche al mio correlatore, l'Ingegnere Claudio Bianchi, per avermi seguito ed assistito durante questi mesi di sviluppo, e per la simpatia dimostrata. Un sentito ringraziamento anche a tutto il personale dei Laboratori di Facoltà e a coloro che durante lo sviluppo di questa tesi mi hanno aiutato.

Un ringraziamento particolare va alla mia famiglia che mi ha sostenuto e motivato in tutti gli anni di studio e soprattutto a mio fratello Davide che mi ha sempre spronato a dare il meglio. Ringrazio la mia ragazza Alessia che ha sopportato la mia passione esagerata per l'aerospazio e le lunghe spiegazioni sul mio lavoro di tesi.

Desidero inoltre ringraziare i compagni di università con i quali ho condiviso il percorso di Laurea Triennale, in particolare Danilo, Giuseppe e Lucia.